DECISION
SCIENCE
CONSORTIUM, INC.

TESTING AND EVALUATING C^{3}I SYSTEMS THAT EMPLOY AI

(CLIN 0001)

VOLUME 4: PUBLISHED ARTICLES

Jacob W. Ulvila, Leonard Adelman, Monica M. Constantine, Paul E. Lehner, and Elissa Gilbert

Decision Science Consortium, Inc.
1895 Preston White Drive, Suite 300
Reston, Virginia 22091

January 1991

Final Report
Period of Performance: 16 September 1988 - 15 September 1990
Contract Number: DAEA18-88-C-0028
PR&C Number: W61DD3-8057-0601
AAP Number: EPG 8048

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

Prepared for:
U.S. Army Electronic Proving Ground
ATTN: STEEP-ET-S (Mr. Robert J. Harder)
Fort Huachuca, Arizona 85613-7110

The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

TECHNICAL REPORT 90-9

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): 90-9	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Decision Science Consortium, Inc. / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: US Army Electronic Proving Ground STEEP-ET-S
6c. ADDRESS (City, State, and ZIP Code): 1895 Preston White Drive, Suite 300 Reston, Virginia 22091	7b. ADDRESS (City, State, and ZIP Code): Ft. Huachuca, Arizona 85613-7110
8a. NAME OF FUNDING/SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL (If applicable): STEEP-ET-S	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAEA-18-88-C-0028
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification): TESTING AND EVALUATING C^3I SYSTEMS THAT EMPLOY AI -- VOLUME 4: PUBLISHED ARTICLES

12. PERSONAL AUTHOR(S): Jacob W. Ulvila, Leonard Adelman, Monica M. Constantine, Paul E. Lehner, and Elissa Gilbert

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical	FROM Sep 88 TO Sep 90	1991 January 31	109

16. SUPPLEMENTARY NOTATION: The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Expert Systems, Testing, Knowledge-Based Systems, Artificial Intelligence, Multiattribute Utility

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This volume contains reprints of published articles produced under this contract. They are entitled:

"Static Analysis Tools for Expert Systems"
"Evaluating Expert System Technology"
"Testing Knowledge-Based Systems: The State of the Practice and Suggestions for Improvement"
"A Note on the Application of Classical Statistics to Evaluating the Knowledge Base of an Expert System"
"Knowledge-Based Systems in the Army: The State of the Practice and Lessons Learned, with Implications for Testing"
"Testing Knowledge-Based Systems"

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Mr. Robert J. Harder	22b. TELEPHONE (Include Area Code): (602) 538-2090	22c. OFFICE SYMBOL: STEEP-ET-S

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED

CONTENTS

STATIC ANALYSIS TOOLS FOR EXPERT SYSTEMS*

Elissa Gilbert

Booz, Allen & Hamilton

1725 Jefferson Davis Highway

Arlington, Virginia 22202

Introduction

The idea of artificial intelligence has existed for decades, even before the term was coined at the Dartmouth conference. This past decade saw this concept begin to be realized, as expert systems emerged as a viable commercial technology. However, although expert systems began to be used commercially, little attention was paid to the issue of testing these systems until recently.

This was a dangerous oversight. Expert systems are also known as rule-based programs, and they are programs. The history of software engineering shows that more than half the cost of software development occurs during testing and maintenance. There is little reason to expect less expense from expert system software. Testing expert system software may be even more expensive since there usually are no requirements or design documents to guide the testing.

There are two kinds of testing that can be used. Dynamic testing consists of running the program on test cases and comparing the output produced with the output expected. Static testing is concerned with examining the underlying structure of the program.

*To appear in *Proceedings of Test Technology Symposium II, 1989.*

Static testing is particularly important for expert systems. Dynamic testing cannot validate the reasoning mechanism, only the results. Because expert systems are usually developed to mimic the thinking of a human expert in an area in which there is no readily determined right or wrong answer, it is not enough that the system produce the right answer; it must produce the right answer in the right way before the system can be trusted. This means that the structure of the rule base, which indicates the reasoning, must be verified. Static analysis provides a way to examine the structure. Since the expert system's output depends on the inference mechanism as well as on the rules, static analysis alone is not sufficient. It should be used as a complement to, not replacement for, dynamic testing.

STATIC ANALYSIS TOOLS FOR TESTING, EVALUATING, AND DEBUGGING EXPERT SYSTEMS

Several static analysis tools have been reported in the literature. This paper discusses the techniques used in four recent tools. CHECK, described in [6], is the earliest of the tools described here. It provides static analysis checks for the Lockheed expert system shell. EVA, the Expert Systems Validation Associate, described in [9] is intended to be used for a variety of expert system shells, such as KEE and ART. The Expert System Parsing Environment, ESPE, is a tool for analyzing expert systems developed with IBM's Expert System Development Environment. It was a research project at R.P.I., reported in [2, 3, 4, 5, 8]. The Expert System Examiner, ESE, is a tool developed by Booz, Allen, & Hamilton while testing expert systems for a government agency. ESE is based on ESPE, and has been used to test three expert systems developed with several different shells.

These four tools, developed independently (except for ESPE and ESE), share much in common. All of these tools either implicitly or explicitly consider an expert system's rule base to be a graph or network. In the graph of a rule base, there are nodes that represent rules and nodes that represent the hypotheses that appear in the rules' premises and conclusions. There is an arc from a hypothesis to each rule whose premise it appears in. There is an arc to a hypothesis from each rule that asserts the hypothesis in its conclusion.

ESPE can plot the graph of the expert system. It can display the rules several ways. The entire rule base can be graphed. Graphs involving only specified nodes can be plotted. It also can plot the subset of the rules that are contained in a specified focus control block (FCB). FCB's in the Expert System Development Environment organize the rules and structure the inference engine's search. FCB's form a tree which can also be graphed.

Given the graph of the expert system, cycles can be detected. Cycles may or may not be an error, depending on the inference engine and the rule base. EVA also checks for overlapping cycles (cycles which share some common rules), which it considers evidence of a poorly-structured knowledge base.

ESPE and ESE count the number of paths between all pairs of nodes in the graph. Each path represents a potential flow of control. ESE's calculation of paths is a little more complicated than ESPE's, because it allow variables in rules. Variable nodes in a rule premise or conclusion duplicate that rule for each possible value of the variable.

In addition to calculating paths, ESPE and ESE allow you to examine path lengths. A long path may indicate tricky or contorted reasoning. A short path may indicate jumping to a conclusion. The user specifies what a long or short path is.

Input and output nodes can also be identified from the graph. Input nodes appear only in the left hand side of rules. Output nodes appear only in the right hand sides of nodes. The meaning of these nodes depends on the inference engine. In some expert system shells, unless these nodes are specified as user-supplied values, or as goals, they are errors, and cannot lead to further rule execution. Some authors refer to these as dead-end or unreachable nodes,because of this. Unexpected values in the lists of input and output nodes reflect incomplete lines of reasoning. Unexpected input values mean that a line of reasoning that should generate that value is missing. Unexpected output values mean that some line of reasoning that should use that value is missing.

Each of the tools checks the expert system rule base for logical inconsistency. There are three kinds of inconsistency that they look for. Redundancy occurs when two rules have the same premise and the same conclusion. Conflict occurs when two rules have the same premise but different conclusions. Subsumption occurs when two values have the same conclusion and one's premise is a subset of the other's. CHECK also looks for unnecessary IF conditions. An unnecessary IF occurs when a condition in one rule's premise conflicts with an IF condition in another rule, all other conditions in the rules' premises are the same, and the rules' conclusions are the same.

The tools identify inconsistency by comparing pairs of rules. Rushby, in [7], points out that this pairwise consistency check is not sufficient, as a sequence of several rules may lead to conflict. EVA has another form of logic checker that may solve that problem. It uses the meta-predicate "incompatible" to discover whether the system can derive inconsistent situations such as A and not A. The real power of this mechanism is that it is not restricted to checking pairs of a condition and its negation; it can check any logical constraint. In the future, when expert systems are developed with requirements specifications that specify illegal situations, this type of logic checker will be a powerful tool in verifying that a system meets its requirements.

EVA's logic checker verifies negative constraints that specify situations that must not arise. There may also be positive constraints, specifying conditions that must be satisfied by facts in the knowledge base. EVA's semantics checker verifies that those constraints are consistent and are satisfied by the rule base.

ESPE and ESE compare pairs of goal (output) values. Either all pairs may be compared, or just a specified pair. Diffences and similarities in the values that lead to these conclusions are reported. For example, consider the rules "if animal_class = mammal and animal_color = tawny and animal_appearance = stripes then animal = tiger" and "if animal_class = mammal and animal_color = tawny and animal_appearance = spots then animal = cheetah". If these are the only two rules that conclude tiger and cheetah, the differences between "animal = tiger" and "animal = cheetah" are "animal_appearance = stripes" and "animal_appearance = spots".

In addition to the actual differences between goal values (information useful for debugging why erroneous conclusions are drawn), the number of differences is an indication of how closely related the values are. In the above example, the conclusions cheetah and tiger differ by only two values, indicating they are closely related. Very closely related pairs should be examined. If it is

important to distinguish between the values, it is important that the values that decide between them will be known with certainty in the environment in which the expert system will be used. If we must be able to tell a cheetah from a tiger, but probably will not know whether it has spots or stripes, this expert system, while technically correct, will not be useful.

When all pairs of output values are compared, ESPE and ESE count how often each non-output value is a difference between a pair of output values. This is a measure of the extent to which the expert system relies on that parameter to decide among possible conclusions. This information is useful in evaluating the expert system. Even if the system is correct, if it relies heavily on information not likely to be known with certainty in your environment, it is not an appropriate system for you to use. In the above example, animal_appearance is used to decide between cheetah and tiger. If animal_appearance is often used to decide between possible output values, but the it will not be known in the environment in which the expert system will be used, or will not be known with certainty, this may be an inappropriate expert system.

ESPE and ESE also compare pairs of input values, either all pairs or specified pairs, identifying differences and similarities in the conclusions they can lead to. This can identify values that should influence the outcome (reach different decisions for different input values), but do not.

ESE can also compare rules that reach the same conclusion. Most likely, these rules should differ only slightly. Rules concluding the same value from vastly different premises should be double-checked.

ESE and ESPE produce a report summarizing the rule base. Both report the number of rules and of values, and identify the input and output values. ESE also calculates the average use of a parameter, and how big the average rule is.

Although most of the techniques implemented by these tools are applicable to any expert system developed with any expert system shell, many shells have unique features which, though powerful, should be closely examined. ESE lists "special" nodes, dependent on the shell. For expert systems developed with Nexpert Object, the report indicates the usage of nodes for strategy, retrieve, execute, write, createobject, deleteobject, show, and reset, and count the number of times they are used. All these nodes affect the state of the expert system without asserting a value, and should be examined.

All the static analysis features described so far examine only the rule base. However, an expert system rule base may be composed of objects as well as rules. The knowledge representation contained in these objects must be verified as well as the reasoning expressed in the rules. ESE is the only tool discussed in this paper which explores the structure of the knowledge representation.

ESE produces a summary report which shows how many classes and objects there are. It lists the average number of classes, subobjects, and properties per object, and the average number of subclasses and properties per class. These numbers give a feel for the level of description associated with each item. For Nexpert Object, the report also lists slots of objects which have IF CHANGE metaslots associated with them; it also sorts and lists object properties by their inference category. Object's contexts can also be reported. IF CHANGE metaslots, inference categories,

and contexts are all mechanisms that affect the state of the system, or its control, outside of the [illegible]s.

ESE allows the tester to identify objects that belong to few or many classes, or have few or many subclasses or properties. The user can also identify classes that have many or few objects in them, many or few subclasses, and many or few properties. The user decides what few and many means; the average numbers in the summary report provide guidance for this. This can identify objects which are poorly represented, or objects which have extraneous detail.

STATIC ANALYSIS TOOLS FOR MAINTAINING EXPERT SYSTEMS

Development of an expert system, or any software, does not end with the test and release of the product. Maintenance and enhancements to the system occur and need to be tested. This is especially true for expert systems, as they are developed via a rapid-prototyping methodology. Rapid prototyping results in several versions of the expert system in quick succession. Booz, Allen's client produces a new prototype approximately every eight weeks. The new prototypes must be tested, both to verify that the old functionality continues to work and to examine the new capabilities.

The static analysis tools described for testing can be used to examine each new release. However, the techniques described above aren't helpful in comparing the two systems, to assess the growth of the system. They don't ease the burden of regression testing. Static comparison of the two rule bases can be useful for this.

ESPE provides tools to help the developer assess the effect of changes before they are made. It requires the user to load the original rule base. The user can then enter any number of new rules, or modify or delete existing rules, and get a report showing the effects of these changes. Effects it looks for are the addition or deletion of nodes or paths, changes in the input or output nodes, changes in path length, inconsistency of new rules and old ones, and changes in the differences between pairs of goal values.

Booz, Allen has implemented a variation of this. Instead of the user interactively entering changes and noting their effects, two complete rule bases are compared. Their differences, resulting from modifications made to refine the prototype, are reported. Identifying no differences between parts of the two systems is as important as identifying differences. No differences indicate no change, and therefore the regression tests need not cover that part of the system.

The comparison of the expert systems first notes changes in the size of the rule base, in both the number of nodes and the number of rules. Changes in the average use of nodes and the average size of rules are reported. It also lists nodes and rules that were added or deleted. Deletions in particular should be verified because they remove knowledge from the system. The comparison checks if any rules in the new rule base are inconsistent with rules in the original. It finds changes in input/output nodes. Paths that have been added to the rule base are reported, as are paths that have been deleted. Pairs of nodes where the length of the longest or shortest path between these nodes has changed are also listed.

The object representation is also compared. Changes in the number of objects and classes are reported, as are changes in the averages of the number of classes objects belong to, number of subobjects objects are composed of, the number of properties describing an object, the number of subclasses a class has, and the number of properties describing a class. It lists objects and classes that were deleted from or added to the system, and identifies objects and classes that have been modified. It also finds properties whose metaslots have changed. Finally, it can identify objects that have changed contexts, and contexts which have different objects in them.

STATIC TESTING IN PRACTICE

While much has been written about these tools, little has been writen about whether they were actually used in testing real expert systems, whether they proved practical and provided benefits to their users.

Nguyen tells us that CHECK "was devised and tested on a wide variety of knowledge bases built with a generic expert system shell." He does not say whether it proved useful to them, whether it found errors that otherwise would not have been caught, and what role it played in their testing process. Although EVA is a comprehensive tool, there is no indication of where or how it is being used. ESPE was developed as a university research project. Though commercially sponsored, there is no indication that its techniques are being used by the sponsor.

One real-world use of a consistency checker was reported by Bachant in [1]. Though an automated rule checker was used, it wasn't found helpful as the level of consistency within the system, XCON, was very low. Bachant says that checking many aspects yielded too many exceptions for the process to be meaningful. This would seem to indicate either a problem with their definition of consistency or with their rule base. It doesn't indicate that static analysis is not worthwhile.

Unlike the other tools reported, Booz, Allen's tools were developed not as research efforts, but as tools to be immediately put to use testing real-world expert systems. They have been used to test three different expert sytems which were developed using three different expert system shells. These were ESL (a Lisp-based expert systems language), a pseudo-English, and Nexpert Object. The tools were able to be used with only minor changes to the component that parses the expert system, indicating the broad applicability of its techniques.

Not all the features of Booz, Allen's tools have been used. One reason for this is that the tools are written in Common Lisp, and the rule bases are large (thousands of rules). These two factors combine to make static analysis a slow process that generates a lot of output. However, the summary reports, path calculation, and consistency checker have been found useful. If rule bases were developed more modularly, the tools could be applied to each module separately, generating results more rapidly, with less output produced.

The consistency checker identified several problems with the rule bases. With large rule bases, these error could not easily be found through tests or through manual examination of the rule base. The strength of the summary report and path calculation is that they retroactively document the system and provide insight into its structure that would otherwise have to come from long,

manual study of the internals of the expert system. Even though input and output nodes appear to be basic information, there is often no documentation indicating what inputs the system expects and what output it is expected to produce. By identifying these nodes, ESE provides a starting point for the testing process.

CONCLUSIONS

Testing all software is a challenge. Testing expert systems is an even greater challenge, due to a lack of requirements and a "hidden" flow of control, as well as the inherent challenge of verifying non-algorithmic processes. Static analysis of an expert system is an important part of its verification. The tools described in this paper can simplify the process and provide valuable information to the system's tester.

REFERENCES

[1] Bachant, Judith, "Validating and Testing XCON," Validation and Testing Knowledge-Based Systems Workshop, St. Paul, Minnesota, August 1988.

[2] Bansal, Rahul, Debugging, Testing, and Maintaining Expert Systems, Master's thesis, Rensselaer Polytechnic Institute, December 1987.

[3] Franklin, W. R., Bansal, Rahul, Gilbert, Elissa, and Shroff, Gautam, "Debugging and Tracing Expert Systems," International Hawaii Conference on System Sciences, January 1988.

[4] Franklin, W. R., Bansal, Rahul, and Gilbert, Elissa, "Sensitivity Analysis of Expert Systems," Validation and Testing Knowledge-Based Systems Workshop, St. Pau, Minnesota, August 1988.

[5] Gilbert, Elissa, Software Tools for the Testing and Maintenance of Expert Systems, Master's thesis, Rensselaer Polytechnic Institute, Decebmer 1987.

[6] Nguyen, Tin A., Perkins, Walton A., Laffey, Thomas J., and Pecora, Deanne, "Knowledge Base Verification", AI Magazine, Summer 1987, pp. 69 - 75.

[7] Rushby, John, Quality Measures and Assurance for AI Software, NASA Contractor Report 4187, Langley Research Center, 1988.

[8] Shroff, Gautam, EXPLOT: A Software Tool for Analyzing Expert Systems, Master's thesis, Rensselaer Polytechnic Institute, May 1987.

[9] Stachowitz, Rolf A., Chang, Chin L., and Combs, Jacqueline B., "Research on Validation of Knowledge-Based Systems," Validation and Testing Knowledge-Based Systems Workshop, St. Paul, Minnesota, August 1988.

EVALUATING EXPERT SYSTEM TECHNOLOGY*

Leonard Adelman and Jacob W. Ulvila

The evaluation of expert systems is becoming increasingly important, for expert systems are moving out of the laboratory and into operational use. How well do these systems work? Do they do what their developers claim? Are their knowledge bases reliable and valid? Do they actually improve operator (and organizational) performance? Can they be effectively integrated with and maintained among more conventional software? [illegible] evaluation methods provide a way of [illegible] questions for the user community.

The goal of this chapter is two-fold. First, we present a general approach and point toward specific methods for testing and evaluating expert systems. Recognizing that experience in software testing has shown that no single method is completely adequate (e.g., Beizer [10]; Bellman and Walter [11]; and Howden [34]), the approach is multi-faceted. It has been successfully used to evaluate expert systems and other forms of advanced information and decision support system (DDS) technology (e.g., see Adelman and Donnell [5]). In addition, we will review different methods for implementing each facet of the approach, and provide a framework for integrating their results (e.g., see Ulvila et al., [64]).

The second goal is to assist in the integration of test and evaluation methods into the design and development process. Test

*In S.J. Andriole and S.M. Halpin (Eds.), *Information Technology for Command and Control*. NY: IEEE Press.

and evaluation represents the control mechanism in software system design and development. This is no less true for expert systems than for more conventional forms of software, for test and evaluation provides the feedback that keeps the development process on track. In fact, it is perhaps even more important for expert systems because their development process emphasizes prototyping. Consequently, this chapter also identifies the applicability of different classes of evaluation methods to different stages in the design and development process.

OVERVIEW

This section presents a blueprint for system development, a multi-faceted evaluation approach, and integrates the two.

System Development Blueprint

Figure 1 presents Andriole's [7] "nine step protoytyping design blueprint" for developing decision support systems (DSSs). We emphasize "DSS" instead of restricting ourselves to expert system technology for three reasons. First, expert systems are most often seen by potential users (and sponsors) as aids to supporting, not replacing human decision makers; for example, see the surveys by Constantine and Ulvila [14] and Mackie and Wylie [43]. Second, expert systems are also often seen by members of the technical, development community as a type of DSS. Mittra [46] has, for example, shown that the components of an expert system correspond quite naturally with the elements of the typical DSS. And, third, the DSS design blueprint is consistant

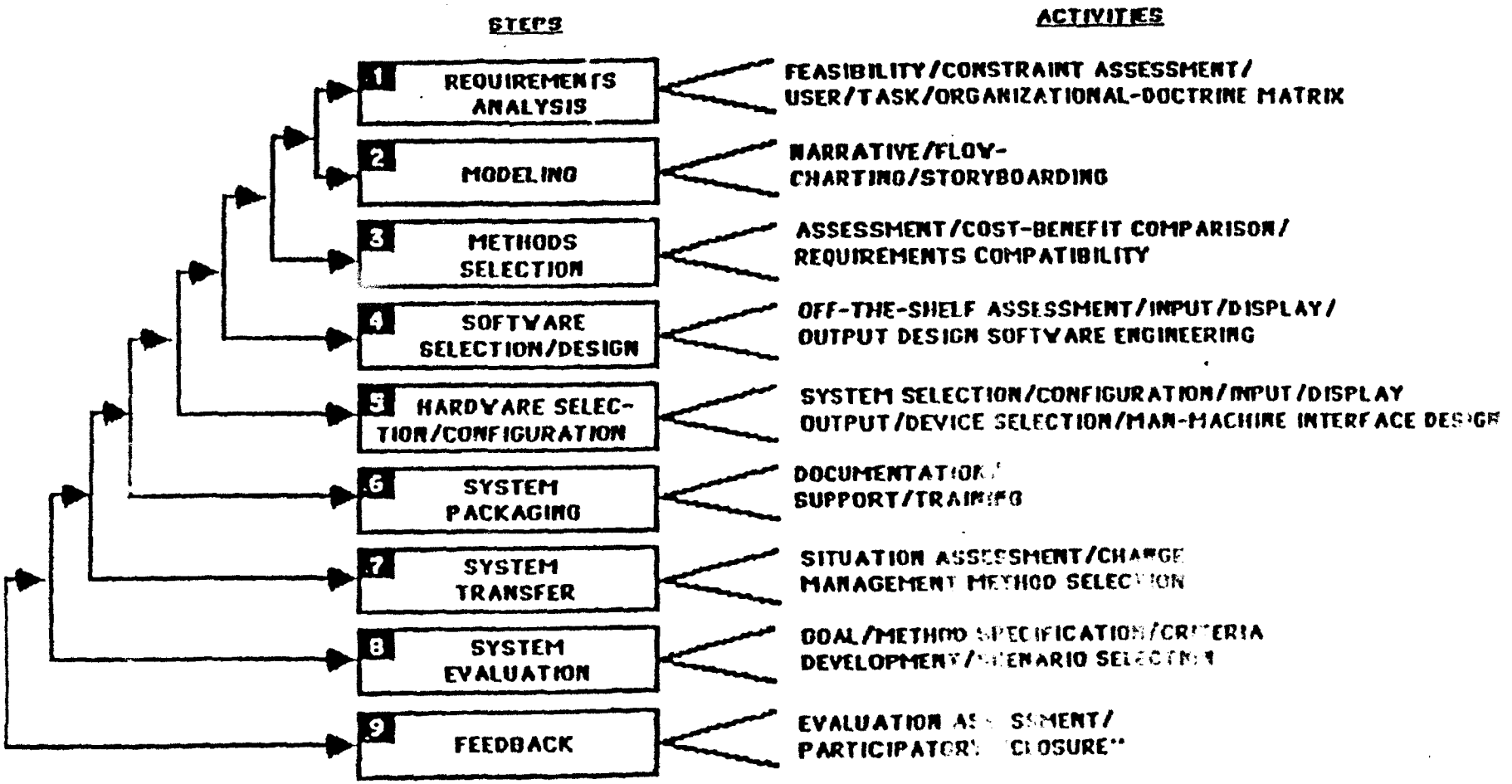


Figure 1

Andriole's Nine-Step Prototyping Design Blueprint

with more traditional software system development frameworks (e.g., see Sage [57]). In particular, it is quite consistent with the system engineering phases in the "knowledge activity matrix" proposed by Rook and Croghan [54] to help structure knowledge acquisition and specification tasks so as to facilitate the transition of expert systems from laboratory to operational settings.

Requirements analysis is the first step, and it has been referred to as "systems targeting" by Andriole [7]. Its goal is to profile the user, the tasks that s/he perform, and the organizational context within which the user and system will operate. Requirements analysis have too often been absent from expert system development efforts. As Culbert et al. [16], Green and Keyes [31], Rook and Croghan [54], and Rushby [55] note, such an omission has significantly impacted the formal testing and evaluation of expert systems. Moreover, it has undoubtedly delayed the transition of some expert systems to operational settings because the user community targeted as its operators have not had a significant role in tailoring the system. Previous research (e.g., Adelman [1]; Huber [35]; Shycon [59]) has documented the importance of user involvement to the successful implementation of advanced information and decision technology. Although the general approach and many of the evaluation methods are still applicable even if a requirements analysis has not been performed, we will assume that it has been performed throughout the remainder of this chapter. This assumption is consistent with our goal of bringing software engineering principles and methods to bear on the expert system development process.

The second step in Andriole's design blueprint is functional modeling. Although there are numerous ways to model a DSS, the goal of each is to show decision makers exactly what the system will be capable of doing, and how it will do it. This is where storyboards (Andriole [8]), which are paper-and-pencil prototypes of the user interface, are particularly important.

The third step is the selection of analytical methods for the DSS's model-based management system. Analytical methods need to be developed consistent with the requirements analysis and functional models. We assume here the selection of expert system technology methods for expository purposes.

The fourth and fifth steps are software and hardware selection, and the subsequent design and development of the DSS. This is where the knowledge elicitation and representation activities that make expert systems so different from other forms of software occurs.

The sixth and seventh steps are system packaging for and transfer to the host organization. The eighth and ninth steps are system evaluation and feedback, respectively. Consistent with a prototyping perspective, Figure 1 emphasizes iteration between the steps in the blueprint. The use of formal evaluation methods at these steps produces data for effective iteration.

A Multi-Faceted Evaluation Approach

Adelman and Donnell [5] presented a three faceted (or phased) approach for evaluating DSSs, and demonstrated its' potential applicability by using it to evaluate an expert system

prototype. The three phase evaluation approach is composed of a subjective phase for obtaining users' opinions regarding the system's strengths and weaknesses; a technical evaluation phase for "looking inside the black box"; and an empirical evaluation phase for assessing the system's impact on performance. Specifically, the subjective evaluation phase focuses on evaluating the DSS from the perspective of potential users. The goal of the subjective evaluation is to assess whether the users like the DSS; what they consider to be its strengths and weaknesses; and what changes they would suggest for improving it.

The technical phase focuses on evaluating the DSS from both an internal (algorithmic and/or heuristic) perspective and an external (systemic input/output) perspective. For example, most people considering the technical evaluation of an expert system might focus on assessing the logical (and functional) adequacy and accuracy of its knowledge base. Rushby [55] has called these "competency requirements." However, from a transfer and maintenance perspective, one also needs to be concerned with conventional test and evaluation issues, such as whether the system can be effectively and efficiently integrated with other software and hardware systems in the operational environment, and whether it was designed consistent with the organization's design and coding standards. Rusby has called these concerns "service requirements." A comprehensive test and evaluation framework needs to address both classes of requirements.

The empirical evaluation phase focuses on obtaining objective measures of the system's performance. The goal of the empirical phase is to assess, for example, whether persons make

significantly better or faster decisions or use significantly more information working with rather than without the system, and to identify mechanisms for improving performance. It is important to note that the potential users of expert system technology may not be experts in the substantive domain. In these cases, one needs both bonified experts and representative users to participate in the evaluation. The experts are needed for the technical evaluation of the knowledge base; the users for the empirical evaluation of system performance. If possible, experts should also participate in the empirical evaluation in order to systematically assess whether system performance is a function of user type.

Integrating the Design Blueprint and the Evaluation Approach

Adelman [2] has classified evaluation methods according to the subjective, technical, and empirical phases of the above evaluation approach. Methods for all three phases are applicable during steps 8 and 9 (evaluation and feedback, respectively) of the development process. In addition, specific methods are more or less applicable to other steps in the development cycle. In particular, subjective evaluation methods tend to be most applicable early in the cycle (steps 1 and 2) because they represent an explicit means for defining the judgments of members of the sponsoring team and potential users of the system. For example, Rockmore et al. [53] used a subjective evaluation method called multi-attribute utility assessment (MAUA), and a MAUA-based cost-benefit analysis, to select among various types of DSS

technology, including expert systems, for subsequent development. Slagle and Wick [60] used a subjective method analogous to MAUA to evaluate candidate expert system application domains. And Bahill et al [9] used MAUA to address the valuative and technical judgments inherent in selecting among expert system shells.

Technical evaluation methods are most applicable in software design and development (step 4). For example, as part of the knowledge elicitation and representation process one should routinely assess the adequacy and accuracy of the knowledge base. This can be accomplished in formal evaluations, as well as through informal ones, by using (1) static testers to help assess the knowledge base's logical consistency and completeness, and (2) experts, both those participating in development and those acting as evaluators, to help assess the knowledge base's functional completeness and predictive accuracy. In addition, traditional software test and verification methods can be used to help assess the "service" versus "competency" requirements of the expert system. These methods have considerable applicability (a) prior to programming code for verifying requirements analyis documentation and functional models of the software (steps 1 and 2), and (b) once the development process is well underway during hardware configuration, system packaging, and system transfer (steps 6 and 7).

In contrast to technical evaluation procedures, which focus on how well the system was developed, empirical evaluation methods focus on how well decision makers can perform the task with (versus without) the system. Remember, the expert system may be addressing only part of a much larger organizational decision.

Even if the technical evaluation of the knowledge base shows that it has perfect predictive accuracy, the expert system's contribution still may not ensure better decision making for the larger decision problem.

From an iterative, prototyping perspective, it is anticipated that experiments will be conducted throughout software development (step 4) and hardware configuration (step 5) as a means of objectively measuring the performance of the system and testing hypotheses for improving it. During this iterative process, experiments also can be used to evaluate system documentation (step 6). Prior to software development, experiments can be used in evaluating alternative storyboards. After transferring the system to the test organization, experiments, quasi-experiments, and case studies can be used to evaluate performance in the operational setting. Finally, there are other empirical evaluation methods. In particular, simulations and statistical analyses of historical data are sometimes applicable during requirements analysis for assessing the potential utility of DSSs using different analytical methods.

The remainder of this chapter will overview subjective, technical, and empirical evaluation methods. A book-length treatment of these methods can be found in Adelman [3].

SUBJECTIVE EVALUATION METHODS

The goal of subjective evaluations is to assess the system from the perspective of potential users and sponsors. This is accomplished by identifying measures of effectiveness (MOEs) that

will provide the information required to assess the system's utility. The explicit identification of MOEs is particularly important at the beginning of the development process because they represent (a) reference points for the development team to use, and (b) criteria for evaluators to monitor in order to assess whether the development process is on track.

Gaschnig et al. [28, p. 258] have emphasized the importance of developing MOEs early in the expert system development process. "It is important for system designers to be clear about the nature of their motivations for building an expert system. The long-range goals must also be outlined explicitly. Thus stage 1 of a system's development, the initial design, should be accompanied by explicit statements of what the measures of the program's success will be and how failure or success will be evaluated. (italics theirs) It is not uncommon for system designers to ignore this issue at the outset, since the initial challenges appear so great upon consideration of the decision-making task that their expert system will have to undertake. If the evaluation stages and long-range goals are explicitly stated, however, they will necessarily have an impact on the early design of the expert system."

Multi-Attribute Utility Assessment

Riedel and Pitz [52, p. 986], as well as others (e.g., Adelman and Donnell [5]; Andriole [7]; Keeney and Raiffa [37]; Ulvila et al. [64]), have pointed out that Multiattribute Utility Assessment (MAUA) "... provides a formal structure for conceptualizing MOE's, a mechanism for both decomposing the

global MOE into its component dimensions and for reintegrating them to yield one summary measure of value." When applying MAUA to the evaluation of expert systems and other types of DSS, the sytem is conceptually decomposed into attributes that can be defined well enough so that one can obtain either subjective or objective measures (MOEs) of how well the system performs on each attribute. This decomposition typically proceeds through the creation of a value hierarchy, such that the global attribute entitled "the overall utility (or value) of the DSS" is decomposed into major categories of attributes, which are further decomposed, and so forth, until one is reasonably confident that one can define and obtain precise, reliable, and valid measures (or scores) of the system on each attribute. Table 1 presents the value hierarchy developed by Adelman and Donnell [5].

Reintegration typically occurs within MAUA through the application of utility functions and relative importance weights. An expert system is usually evaluated on many different attributes, all of which need to be defined as precisely as possible. The natural measurement scale for an attribute depends on the nature of the attribute. For example, the scale for an attribute could be in objective units (e.g., minutes for time) or subjective units (e.g., answers on a questionnaire) depending on the attribute. Nevertheless, a common scale is required to compare scores on one attribute with scores on another, that is, "apples with oranges," and, by so doing, obtain an overall score for the system. A utility scale, which conceptually measures psychological value or satisfaction, meets this requirement.

0.0 OVERALL UTILITY

- 1.0 Aid/User Interface
  - 1.1 Match with Personnel
    - 1.1.1 Training & Technical Background
    - 1.1.2 Work Style, Workload & Interest
    - 1.1.3 Operational Needs
  - 1.2 Aid Characteristics
    - 1.2.1 General
      - 1.2.1.1 Ease of Use
      - 1.2.1.2 Understanding Aiding Process
      - 1.2.1.3 Ease of Training
      - 1.2.1.4 Response Time
    - 1.2.2 Specific
      - 1.2.2.1 User Interface
      - 1.2.2.2 Completeness of Data Files
      - 1.2.2.3 Accuracy of Expert Judgments
      - 1.2.2.4 Ability to Modify Judgments
      - 1.2.2.5 Understanding of Aid's Algorithms
      - 1.2.2.6 Utility of Graphs
      - 1.2.2.7 Utility of Printouts
      - 1.2.2.8 Understanding of Text
- 2.0 User/Aid Organization
  - 2.1 Efficiency Factors
    - 2.1.1 Acceptability of Time for:
      - 2.1.1.1 Task Accomplishment
      - 2.1.1.2 Data Management
      - 2.1.1.3 Set-Up Requirements
    - 2.1.2 Perceived Reliability Under Average Battle Conditions
      - 2.1.2.1 Skill Availability
      - 2.1.2.2 Hardware Availability
  - 2.2 Match with Organizational Factors
    - 2.2.1 Effect on Organizational Procedure and Structure
    - 2.2.2 Effect on Other People's Position in the Organization
      - 2.2.2.1 Political Acceptability
      - 2.2.2.2 Other People's Workload
    - 2.2.3 Effect on Information Flow
    - 2.2.4 Side Effects
      - 2.2.4.1 Value in Performing Other Tasks
      - 2.2.4.2 Training Value
- 3.0 Organization/Environment
  - 3.1 Decision Accuracy
  - 3.2 Match Between Aid's Technical Approach and Problem's Requirements
  - 3.3 Decision Process Quality
    - 3.3.1 Quality of Framework for Incorporating Judgment
    - 3.3.2 Range of Alternatives
    - 3.3.3 Range of Objectives
    - 3.3.4 Weighting of Consequences
    - 3.3.5 Assessment of Consequences of Alternatives
    - 3.3.6 Re-examination of decision-making process
    - 3.3.7 Use of Information
    - 3.3.8 Consideration of Implementation and Contingency Plans
    - 3.3.9 Effect on Group Discussions
    - 3.3.10 Effect on Decision-makers' Confidence

Table 1

Adelman and Donnell's MAUA DSS Evaluation Hierarchy

Utility (or value) functions are used to translate system performance on an attribute into a utility score on that attribute. Then, relative importance weights (or other forms of decision rules) are used to assess the relative value of a utility score on one attribute with the utility score on another and, thereby, obtain an overall utility score for the system. (This weighting procedure is formally valid if additivity assumptions are met; see Keeney and Raiffa [37]. An assumption of additivity is generally a reasonable approximation; see Edwards [24].)

Other Subjective Evaluation Methods

There are other subjective evaluation methods besides MAUA. For example, Adelman [3] also discusses traditional cost-benefit analysis, the dollar-equivalent technique, decision analysis, and a MAUA-based cost-benefit analysis. In addition, Liebowitz [42] has used the Analytical Hierarchy Process developed by Saaty [56], Tong et al. [63] have proposed a frame-based approach, and Klein and Brezovic [39] and Slagle and Wick [60] have used subjective evaluation approaches analogous to MAUA. It is important to emphasize that in all cases these methods use personal judgments. This initially might be disturbing to (and difficult for) members of both the sponsoring and developing teams, for it emphasizes the subjective process decision makers go through when evaluating DSSs. To quote Riedel and Pitz [52, pp. 987-988], "There is no way to avoid the fact that the overall MOE must be based on such judgments, or the fact that no mechanical procedure can replace this subjective assessment...." Through the explicit identification of MOEs and procedures for

converting performance scores into a global MOE, subjective evaluation methods provide refencence points for the development team to use when developing the expert system, and criteria for the evaluator to monitor in order to assess whether the development process is on track.

TECHNICAL EVALUATION METHODS

Three classes of technical evaluation methods are, in turn, briefly overviewed in this chapter: (a) static testing for assessing the logical consistency and adequacy of the knowledge base; (b) using domain experts for assessing the functional completeness and predictive accuracy of the knowledge base; and (c) conventional software test and verification methods for assessing the service requirements of the entire system.

Logical Consistency and Completeness

As Rusby [55] points out, the concepts of static testing in conventional software testing can be readily extended to expert systems because, in both instances, the focus is on detecting anomalies in the program without actually executing it on test cases. To quote Rushby (p. 92), "An anomaly in a program is *nothing more* than an apparent conflict between one indication of intent or purpose and another ...." The types of anomalies of particular interest in expert systems pertain to the logical consistency and logical completeness of the knowledge base.

Researchers (e.g., Kirk and Murray [38], Nazareth [47], and Rushby [55]) have developed taxonomies of anomalies in the

knowledge base that are amenable to static testing. Some of these anomalies are listed below. In doing so, we assume that the knowledge base is represented in the form of "if-then" production rules or can be transformed into such a representation. As Nazareth [47] points out, "For systems that employ more involved representation schemes, the nature of the verification task may differ" (p. 257).

- Redundant Rules. Individual rules or groups of rules that essentially have the same conditions and conclusions.
- Subsumed Rules. When one rule's (or rule group's) meaning is already expressed in another's that reaches the same conclusion from similar but less restrictive conditions.
- Conflicting Rules. Rules (or groups of rules) that use the same (or very similar) conditions, but result in different conclusions, or rules whose combination violates principles of logic (e.g., transitivity).
- Circular Rules. Rules that lead one back to an initial (or intermediate) condition(s) instead of a conclusion.
- Unnecessary If Conditions. The value on a condition does not affect the conclusion of any rule.
- Unreferenced Attribute Values. Values on a condition that are not defined; consequently, their occurrence cannot result in a conclusion.
- Illegal Attribute Values. Values on a condition that are outside the acceptable set of values for that condition.
- Unreachable Conclusion (and Dead Ends). Rules that do not connect input conditions with output conclusions.

Static testing for the above anomalies could be performed

manually for small, well structured knowledge bases. For even moderately sized knowledge bases, however, this approach is precluded by the amount of effort required and the probability of disagreements among testers. Consequently, researchers (e.g., Culbert and Savely [17]; Franklin et al. [27]; Nguyen et al. [48]; Stachowitz et al [61]) have begun developing automated static testers. We do not have the space here to discuss these different efforts. However, we do want to caution the reader that automated static testers are not without their limitations. To quote Nazareth [47], "In most cases the verification process is closely dependent on the structure of the problem domain, making translation of principles to other systems difficult. Additionally, only a subset of the errors identified [above] are covered. ... The expansion of verification scope has serious implications for detection. ... [And] the majority are directed toward applications without uncertain inference" (pp. 265 & 266). Nevertheless, automated static testers represent a major step forward in assessing the logical consistency and completeness of a knowledge base.

Functional Completeness and Predictive Accuracy

By functional completeness we mean to address the range of domain-oriented questions, such as whether the knowledge base contains all desired input conditions and output conclusions, or even "knows" its knowledge limitations. Some of these questions can be answered by domain references. However, the level of domain expertise typically desired for expert systems is

typically not codified in such references. Indeed, Davis [18] has argued that one of the major contributions of expert system technology is the organization and codification impacts it has on various disciplines. Consequently, domain experts are typically required to evaluate the functional completeness of the system. However, one should remember that the system's level of functional completeness depends on its stage of development and, most importantly, the domain requirements resulting from the requirements analysis (step 1).

The predictive accuracy of the knowledge base pertains to the correctness by which the rules (or whatever representation scheme) relates input conditions to output conclusions. Such an assessment is essential for expert systems, for "garbage in" is literally "garbage out." Moreover, domain experts, knowledge engineers, representation schemes, and elicitation methods all potentially represent threats to knowledge base validity during development (Adelman, [4]). Experts, both those who participated in development and those acting as independent evaluators, are typically used to evaluate the predictive accuracy and thus, adequacy of the knowledge base. Expert evaluation typically proceeds in two ways: through examination of the knowledge base, and the evaluation of test cases.

Expert examination of the knowledge base typically focuses on whether the system exhibits "correct reasoning." The obvious concern is, of course, that the knowledge base not have mistakes. However, another concern, and one which Gaschnig et al. [28] pointed out is not shared by all developers, is whether their programs reach decisions like human experts do. Many

psychologists have long argued that this concern can not be answered for one can not, so to speak, look inside an expert's head to obtain the "correct reasoning." Instead, all one can do is build "paramorphic models" (Hoffman, [33]) of the reasoning *process*, and evaluate their predictive accuracy against test cases. Indeed, researchers (e.g., Dawes and Corrigan [19]; Einhorn and Hogarth [25]; Levi [41]; Stewart et al. [62]) have shown that simple linear models can often result in prediction as good as that achieved by experts or the far more complex models found in expert systems.

This is not a resolved issue. As Gaschnig et al. (p. 255) point out, "... there is an increasing realization that expert-level performance may require heightened attention to the mechanisms by which human experts actually solve the problems for which the expert systems are typically built." In addition, Adelman, Rook, and Lehner [6] found that domain experts' judgments of the utility of DSS (including expert system) prototypes was significantly affected by the match between how they and the system attempted to solve the problem. This suggests that, at a minimum, the system's representation and presentation scheme needs to be reviewed.

The predictive accuracy of the knowledge base is performed using test cases and performance standards. The desired standard is ground truth; that is, the correct answers to the test cases. Correct answers are most desirable because substantial research (e.g., see Ebert and Kruse [23]; Goldberg [30] ; Yu et al. [67]) has shown that experts do not always make perfect inferences and,

in fact, often disagree with one another in the kinds of complex domains for which many expert systems are developed. Often, it is inappropriate to expect better predictive accuracy from the system than the expert. (This may not be the case where the system incorporates knowledge from a limited, well defined domain--such as a procedure manual-- or where the system represents the expertise of several experts. Here, it may be appropriate to expect the system to be more accurate than any given expert.)

If ground truth measures exist, one should discriminate between "accuracy" and "bias" in a signal detection sense (Lenner [40]). Accuracy refers to the degree of overlap in the distributions of belief values when the hypothesis is true versus false. Bias refers to the proportion of false negatives (hypothesis true, but user says false) to false positives (hypothesis false, but user says true).

If the correct answers do not exist or, for whatever reason, are inappropriate for the test cases, then one must rely on the judgment of an expert or the consensus judgments of a group of experts. Considerable care must be given to structuring the experts' activities. In particular, the evaluation team must ensure that the experts are "blind" as to whether the system or other experts generated the conclusions to the test cases. This is typically referred to as a "Turing test" (e.g, see Rushby [55]).

In closing this subsection, it is important to note that test case construction is an important issue. To quote Okeefe et al. [49, p. 83], "The issue is not the number of test cases, it

is the coverage of test cases--that is, how well they reflect the input domain. The input domain is the population of permissible input..." (italics theirs). The required coverage capabilities is clearly a statement that needs to be a result of the requirements analysis. For as O'keefe et al. point out, developers frequently devote a disproportionate amount of time to attempting to ensure that the system can handle the truely "expert" cases that may occur very infrequently. Moreover, these "infrequent" cases often become the test cases. This may or may not be appropriate depending on the requirements for the system, and it can certainly be expensive.

An alternative identified by O'keefe et al. is to randomly select test cases using a stratified sampling scheme such that the relative frequency of the cases is representative of those in the operational environment or stipulated in the requirements. Additionally, test cases should be chosen to cover situations where a failure in the system would be especially serious. It is also important that some of the test cases simulate the most common operation of the system.

Service Requirements

Verification testing should be systematically performed for the service requirements of expert systems, just like any other software product. Fagan and Miller [as reported in DeMillo et al. [20] have identified four phases for software testing. The first phase is manual analysis in which the requirements specification and design and implementation plan are analyzed for problems by

experienced software engineers. The second phase is static analysis, which may be manual or automated, in which requirements and design documents and software are analyzed, but without code execution. The third phase is dynamic analysis in which software is executed with a set of test data, such as in random testing, functional testing, and path testing. The fourth phase, which Fagan and Miller consider to be optional, is attempting to prove the program as being correct, such as in mathematical verification. Detailed discussions of these and other methods can be found in, for example, DeMillo et al. [20], Fairley [26], Pressman [51], and Rushby [55].

Section Summary

In closing, we want to make four points about technical evaluation methods. First, as Hamlet [32, p. 666] points out, each method has its strengths and weaknesses and therefore, represent "imperfect test methods." Therefore, evaluators need to use multiple methods to obtain accurate feedback. Second, the intent of testing is to find errors. As Fairley [26, p. 268] points out, "... one has most confidence in programs with no detected bugs after thorough testing and least confidence in a program with a long history of fixes." Third, the best way to minimize the number of errors and the amount of time, effort, and money required to fix them, is to eliminate errors early in development. Consequently, as Gelperin and Hetzel [29] point out, software development life cycles are becoming "preventive" through the application of software testing methods early in the development process. And, fourth, testing methods using experts

to evaluate the knowledge base rely heavily on empirical analysis via test data. However, the reader should keep a clear distinction between the empirical results of technical and empirical evaluation methods. The former focus on how well the expert system's knowledge base was developed; the latter focus on how well system users, who may not be experts, can perform the task with versus without the expert system.

EMPIRICAL EVALUATION METHODS

Empirical evaluation methods can be classified into experiments, quasi-experiments, case studies, simulations, and statistical analyses of historical data (e.g., see Adelman [3]). Only the first two methods are considered here.

Experiments

Experiments are, by far, the most common and commonly thought of empirical evaluation method. Moreover, they are particularly appropriate when a number of people would actually use the developed expert system, for experiments are designed to help generalize from a test sample to the larger population.

One typically thinks of two kinds of experiments. The first kind tests the system against objective benchmarks that represent performance constraints. If the system passes the benchmarks, it proceeds further; if it fails, it undergoes further development or is set aside. "For example, it is not enough to know that with the aid the user can arrive at a decision in 30 min. If the organizational user required a decision in 30 min, the aid would

be effective. If a decision was needed in 15 min, the aid would not be effective" (Riedel and Pitz [52, pp. 984-985]).

It should be noted that such performance benchmarks differ from the more traditional time and efficiency measures used to benchmark computer systems. (Note: Readers interested in the latter are referred to Press [50], who benchmarcked different expert systems on the time required to load and execute different types of knowledge bases, and the amount of disk space required in source and fast-load formats.) Both classes of benchmarks typically get developed during requirements analyses emphasizing a features-based approach. Although such performance constraints may be necessary in real-time, life-critical activities, they are unnecessary for many expert system applications.

Second, performance benchmarks represent non-compensatory decision rules; that is, the system's other features do not compensate for failing the performance benchmark. Such a position may be inconsistent with the compensatory decision rule guiding the sponsoring team's intuitive decision making processes or more formal subjective methods, such as MAUA. After all, it's quite possible that the sponsoring team might be willing to give up some time for task accomplishment (or some whatever) in order to gain even alittle improvement on other MOEs, such as decision performance or personnel staffing requirements (or whatever).

The second kind of experiment, and the one that is focused on here, is a factorial design (e.g., see Cochran and Cox [13]) where (a) one or more factors are systematically varied as the independent variable(s), and (b) the dependent variable(s) are quantitative, objective measures of system performance. There are

five basic components of factorial experiments. First, there are the participants in the experiment. These may or may not be experts depending on the targeted users of the expert system's advice. We focus on "users" because the system operators may or may not be the actual decision makers.

Second, there is the task(s) that the participants are to perform during the course of the experiment. Test cases are often embedded in larger scenarios representative of the organization's problem solving environment in order to effectively assess (1) the users' ability to solve problems with and without the system, and (2) their opinion of system characteristics, such as its speed, explanation capabilities, organizational fit, etc. Remember, the expert system may be addressing only part of a much larger organizational decision.

Third, there is the experimental condition(s) or independent variable(s) of interest, such as whether the participants perform the task with versus without the expert system. The level of task difficulty should be either as representative of the operational environment as possible or matched to the required performance capabilities of the system. The capabilities of the system depend on it stage of development (e.g., see Gaschnig et al., [28]; Marcot, [44]).

Fourth, there is the dependent variable(s) (or MOEs) of interest. Objective measures (e.g., performance and speed), observational measures (e.g., regarding how the system is used) and subjective measures (e.g., user confidence in the solution) can all be used as the dependent variable(s). In the case of

decision quality, one should use either ground truth measures (i.e., the correct answer) for the task or, if they do not exist or are inappropriate, the collective judgment of two or more experts given the large amount of research showing expert disagreement. (The use of one expert is acceptable if the requirement is that the expert system emulate the judgments of that expert.) If ground truth measures exist, one should discriminate between "accuracy" and "bias" in a signal detection sense, as was done for the knowledge base. If experts are used, "blind" ratings as to which experimental conditions produced the solutions are again required to control against bias.

And, fifth, there are the procedures governing the overall implementation of the experiment. Substantial care should be directed toward accurately representing the unaided as well as aided condition to ensure a fair test. If performance is better in the "aided" condition, we want to be able to say that it is due to the expert system and not some other extraneous factor. In order to do so, we need to (ideally) try to control for all "plausible rival hypotheses" (Campbell and Stanley, [12, p. 36]) that might explain the obtained findings. Toward that goal we introduce the concepts of reliability and validity.

Yin [66, p. 36] defines reliability as "demonstrating that the operations of a study--such as the data collection procedures--can be repeated, with the same results." The key concept is replication. In contrast, "valid" is defined by Webster's dictionary [65 , p. 1608] as that which is sound because it is "well grounded on principles or evidence." If an experiment is valid, its conclusions can be accepted; that is,

rival hypotheses have been controlled for.

An experiment can be reliable, but its conclusions invalid. However, an experiment can not be valid if it is unreliable; that is, one can not conclude that the results are well grounded if the evidence upon which they are based is undependable. The basis for good experimentation is, therefore, reliable (i.e., dependable) procedures and measures. Although far from trivial, reliability is typically possible in experimentation because of high experimenter control. For example, the experimenter can pilot-test and subsequently modify the procedures and measures until they produce the same results when applied to the same situation, regardless of who performs the experiment.

We consider four types of validity. First, Yin [66, p. 36] has defined internal validity as "establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships." As Cook and Campbell [15, p. 38] note, "Internal validity has nothing to do with the abstract labeling of a presumed cause or effect; rather, it deals with the relationship between the research operations irrespective of what they theoretically represent" (itallics theirs). Although there are numerous threats to internal validity, randomization of participants to experimental conditions is the most effective means for guarding against them.

In addition, one needs to consider the experiment's construct validity, its statistical conclusion validity, and its external validity. Yin [66, p. 36] has defined construct validity

as "... establishing good operational measures for the concepts being studied." Construct validity is required in order to "make generalizations about higher-order constructs from research operations" (Cook and Campbell [15, p. 38) in a particular study. Good construct validity means that we are measuring that, and only that which we want to be measuring. Of particular concern in expert system evaluations is that the "system treatment" is not confounded by something else. If confounding exists, then the "something else" represents rival hypotheses that could explain our obtained results.

"Statistical conclusion validity is concerned not with sources of systematic bias but with sources of random error and with the appropriate use of statistics and statistical tests" (Cook and Campbell [15, p. 80]). The former concern is with whether the study is sensitive enough to permit reasonable statements regarding the covariation between the independent and dependent variables. The latter concern is with what constitutes appropriate statistical tests of these statements; in this regard, see O'Keefe et al. [49] for expert systems.

As Campbell and Stanley [12 , p. 5] point out, "External validity asks the question of generalizability: To what populations, settings, treatment variables, and measurement variables can this effect be generalized?" (italics theirs). Within the context of expert system evaluations, external validity deals with the extent to which the results of an experiment conducted in a simulated (laboratory) setting will generalize to an operational environment. Consistent with an iterative, prototyping approach, the representativeness of the

experimental setting and the level of the system's performance requirements should advance throughout the development cycle. Although the latter is routinely acknowledged, the former is not. It must be remembered that most information and decision technology fails to be successfully implemented for organizational, not technical reasons. Consequently, increasing the fidelity of the organizational and environmental interfaces between the system and its users is essential in generalizing the performance results obtained in the laboratory to the real world.

Quasi-Experiments

Ideally, field experimentation would be used to assess if the expert system significantly improved performance in an actual organizational setting. For example, appropriate organizational units (e.g., sections in a company or governmental agency) would be randomly assigned to the "with system" and "without system" conditions, and their performance measured until it stabilized. If possible, a "placebo" condition would be included too. Organizational units in this condition would be given some "treatment" that was not hypothesized to have any effect on performance. This is analogous to giving patients sugar pills when evaluating new drugs, and is oriented to controlling for the "Hawthorne effect" (e.g., see Schein [58]) confounding in the "with system" condition that is the result of being given special treatment and not the technology. The unit of analysis is the performance of the organizational unit; consequently, a large enough sample of units would be required for performing

statistical tests.

The sample size and randomization requirements of true experiments is typically *not possible* in many organizations. Quasi-experimental designs should be used in such situations. To quote Campbell and Stanley [12, p. 34], "There are many social settings in which the research person can introduce something like experimental design into his scheduling of data collection procedures (e.g., the when and to whom of measurement), even though he lacks the full control over the scheduling of experimental stimuli (the when and to whom of exposure and the ability to randomize exposures) which make a true experiment possible. Collectively, such situations can be regarded as quasi-experimental designs." [italics theirs]

There are a number of different types of quasi-experimental designs. Campbell and Stanley [12], for example, identify ten types. These include, for example, (a) time series designs, where the organizational unit would be measured for a long period of time before and after receiving the system; (b) multiple time series designs that do not use randomization, but do use a control group that does not receive the system; and (c) nonequivalent (and nonrandomized) control group designs that rely on analysis or covariance to assess whether the pretest and posttest difference for the expert system group is significantly better than that of the control group.

SUMMARY

This chapter has presented a multi-faceted approach to testing and evaluating expert systems. This approach is composed

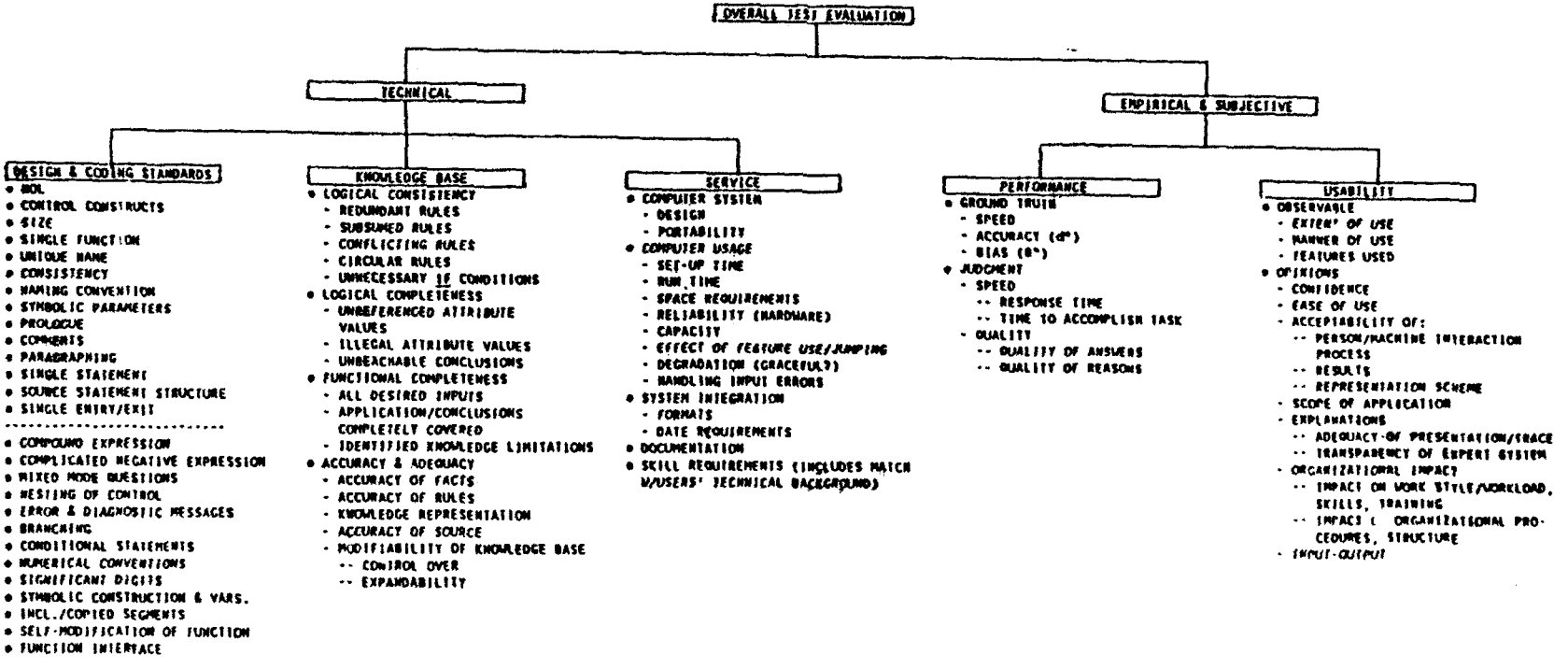
of subjective, technical, and evaluation methods. These methods can be used to evaluate the system at the end of development or used, as we would recommend, throughout the development process to provide feedback that keeps development on track.

The different methods overviewed herein address different test and evaluation criteria. Figure 2 presents a framework that not only summarizes these criteria, but attempts to integrate them by using a Multi-Attribute Utility Assessment (MAUA) hierarchy. This hierarchy builds on the previous work of Ulvila et al. [64]. It can be used in conjunction with MAUA scoring and weighting procedures to assess the overall utility of an expert system to users and sponsors. This is the top-level of the hierarchy. The goal of developers and sponsors of expert systems for operational settings is, of course, the creation of high utility technology.

The hierarchy has two branches. The first contains criteria for technical evaluations. These include design and coding standards (those shown are adaptations of the relevant standards from DOD-STD-1679A [21]; DOD-STD-2167 [22]; JCMPOINST 8020.1 [36]; and MIL-STD-1679 [45];) and competency (i.e., knowledge base) and service (i.e., conventional software) requirements. The knowledge base is decomposed into logical consistency and completeness, functional completeness, and predictive accuracy.

The second branch contains criteria appropriate for the empirical and subjective evaluations. These are grouped according to performance and usability criteria. Performance is decomposed into criteria based on ground truth (or experts' ratings), and

Figure 2: A MAUA Framework for Integrating Test and Evaluation Criteria



the judgments of users. Usability is decomposed into criteria based on the observation of participants working with the system and their judgments of it.

We hope this chapter has made clear that methods exist for assessing expert systems on each of these criteria. That is not to say that no further development work is required, for much research still remains. But it is to say that the test and evaluation community is beginning to assemble the rigorous procedures and technology required to effectively evaluate expert systems. Moreover, subjective evaluation methods like MAU represent a mechanism for converting the individual assessments on these diverse criteria into an overall utility measure. Further research is also required here, for there is little experience converting such assessments into measures of utility. Nevertheless, a firm foundation exists upon which to build.

FOOTNOTES

This paper is based on research efforts performed under (1) a subcontract to George Mason University from Science Applications International Corporation (SAIC) under a contract from the Ft. Leavenworth Field Office of the U.S. Army Research Institute for the Behavioral and Social Sciences (ARI), and (2) a contract to Decision Science Consortium, Inc. from the U.S. Army Electronic Proving Ground (USAEPG). The authors would like to thank Dr. Stan Halpin of ARI, Mr. Paul McKeown of SAIC, and Mr. Robert Harder of EPG for their support. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

Dr. Leonard Adelman is an associate professor in the Department of Information Systems and Systems Engineering at George Mason University, Fairfax, VA, and a Senior Principal Associate at Decision Science Consortium, Inc., in Reston, VA.

Dr. Jacob Ulvila is Executive Vice President at Decision Science Consortium, Inc., Reston, VA.

REFERENCES

[1] L. Adelman, "Involving users in the design of decision-analytic aids: The principal factor in successfulimplementation," Journal of the Operational Research Society, vol. 33, pp. 333-342, 1982.

[2] L. Adelman, "Evaluating decision support systems: Toward integrating evaluation methods into the development process," in Advanced Technologies For Command and Control System Engineering, S.J. Andriole, Ed. Fairfax, VA: AFCEA International Press, in press.

[3] L. Adelman, Evaluating Decision Support Systems. Wellesley, MA: QED Information Sciences, in press.

[4] L. Adelman, "Measurement issues in knowledge engineering," IEEE Transactions on Systems, Man, and Cybernetics, in press.

[5] L. Adelman and M.L. Donnell, "Evaluating Decision Support Systems: A General Framework and Case Study," in Microcomputer Decision Support Systems: Design, Implementation, and Evaluation, S.J. Andriole, Ed. Wellesley, MA: QED Information Sciences, 1986.

[6] L. Adelman, F.W. Rook, and P.E. Lehner, "User and R&D specialist evaluation of decision support systems: Development of a questionnaire and empirical results," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC 15, pp. 334-342, 1985.

[7] S.J. Andriole, Handbook for the Design, Development, Evaluation, and Application of Interactive Decision Support Systems. Princeton, NJ: Petrocelli, 1989.

[8] S.J. Andriole, Storyboard Prototyping for Systems Design: A New Approach to User Requirements Validation and System Sizing. Wellesley, MA: QED information Sciences, 1989.

[9] A.T. Bahill, P.N. Harris, and E. Senn, "Lessons learned building expert systems," AI Expert, vol. 3, pp. 36-45, 1988.

[10] B. Beizer, Software System Testing and Quality Assurance. New York: Van Nostrand Reinhold, 1984.

[11] K.L. Bellman and D.O. Walter, "Analyzing and correcting knowledge-based systems requires explicit models," Proceedings of the AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems, St. Paul, MN, Aug. 20 1988.

[12] D.T. Campbell and J.C. Stanley, Experimental and Quasi-Experimental Designs for Research, Rand McNally, 1966.

[13] W.G. Cochran and G.M. Cox, Experimental Designs (2nd ed.). New York: John Wiley & Sons, 1957.

[14] M.M. Constantine and J.W. Ulvila, "Knowledge-based systems in the Army: The state of the practice and lessons learned," Proceedings of IJCAI-89 Workshop on Verification, Validation and Testing of Knowledge-Based Systems, Detroit, MI, Aug. 17 1989.

[15] T.D. Cook and D.T. Campbell, Quasi-Experimentation: Design & Analysis Issues for Field Settings. Chicago, IL: Rand McNally, 1979.

[16] C. Culbert, G. Riley and R.T. Savely, "Approaches to the verification of rule-based expert systems," Proceedings of SOAR'87: Space Operations-Automation and Robotics Conference, Houston, TX, August 1987.

[17] C. Culbert and R.T. Savely, "Expert system verification and validation," Proceedings of AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems, St. Paul, MN, Aug. 20, 1988.

[18] R. Davis, "Expert systems: How far can they go?" AI Magazine, vol. 10, pp. 65-77, 1989.

[19] R.M. Dawes and B. Corrigan, "Linear models in decision making," Psychological Bulletin, vol. 81, pp. 97-106, 1975.

[20] R.A. DeMillo, W.M. McCracken, R.J. Martin, and J.F. Passafiume, Software Testing and Evaluation, The Benjamin/Cummings Publishing Co., 1987.

[21] DOD-STD-1679A: Software Development (Section 5.3, Programing Standards), Feb. 1983.

[22] DOD-STD-2167: Defense System Software Development (Section 30.3, Detailed Requirements section of General Design and Coding Standards), 4 June 1985.

[23] R.J. Ebert and T.E. Kruse, "Bootstrapping the security analyst," Journal of Applied Psychology, vol. 63, pp. 110-119, 1978.

[24] W. Edwards, " Use of multiattribute utility measurement for social decisions," in Conflicting Objectives in Decisions, D.E. Bell, R.L. Keeney, and H. Raiffa, Eds. New York: John Wiley & Sons, 1977.

[25] H.J. Einhorn and R.M. Hogarth, "Unit weighting schemes of decision making," Organizational Behavior and Human Performance, vol. 13, pp. 171-192, 1975.

[26] R.E. Fairley, Software Engineering Concepts, McGraw-Hill, 1985.

[27] W.R. Franklin, R. Bansal, E. Gilbert, and G. Shroff, "Debugging and tracing expert systems," International Hawaii Conference on System Sciences, January 1988.

[28] J. Gaschnig, P. Klahr, H. Pople, E. Shortliffe, and A. Terry, "Evaluation of expert systems: Issues and case studies," in Building Expert Systems, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, Eds. Reading, MA: Addison-Wesley, 1983.

[29] D. Gelperin, and B. Hetzel, "The growth of software testing," Communications of the ACM, vol. 31, pp. 687-695, 1988.

[30] L.R. Goldberg, "Man versus model of man: A rationale, plus some evidence, for a method of improving clinical inferences," Psychological Bulletin, vol. 73, pp. 422-432, 1970.

[31] J.R. Green and M.M. Keyes, "Verification and validation of expert systems," Proceedings, Western Conference on Expert Systems, pp. 38-43, IEEE Computer Society, Anaheim, CA, June 1987.

[32] R. Hamlet, "Special section on software testing," Communications of the ACM, vol. 31, pp. 662-667, 1988.

[33] P.J. Hoffman, "The paramorphic representation of human judgment," Psychological Bulletin, vol. 57, pp. 116-131, 1960.

[34] W.E. Howden, "Functional program testing," Proceedings of IEEE Compsac, pp. 321-325, 1978.

[35] G.P. Huber, "The decision-making paradigm of organizational design," Management Science, vol. 32, pp. 572-589, 1986.

[36] JCMPOINST 8020.1: Independent Software Nuclear Safety Analysis (Change 2, Appendix F, Section 3.6(3), Specifications and Design Audit and Analysis), 3 March 1984.

[37] R.L. Keeney and H. Raiffa, Decisions With Multiple Objectives. New York: John Wiley & Sons, 1976.

[38] D.B. Kirk and A.E. Murray, Verification and Validation of Expert Systems For Nuclear Power Applications. McLean, VA: Science Applications International Corporation, 1988.

[39] G. Klein and C. Brezovic, "Evaluation of expert systems," in Defense Applications of AI, S.J. Andriole, Ed. Lexington Books, 1989.

[40] P.E. Lehner, "Toward a mathematics for evaluating the knowledge base of an expert system," IEEE Transactions on Systems, Man, and Cybernetics, in press.

[41] K. Levi, "Expert systems should be more accurate than human experts: Lessons from human judgment and decision making," IEEE Transactions on Systems, Man, and Cybernetics, in press.

[42] J. Liebowitz, "Useful approach for evaluating expert systems," Expert Systems, vol. 3, pp. 86-96, 1986.

[43] R.R. Machie and C.D. Wylie, Technology Transfer and Artificial Intelligence. Goleta, CA: Essex Corp., 1985.

[44] B. Marcot, "Testing Your Knowledge Base," AI Expert, vol. 2, pp. 42-47, 1987.

[45] MIL-STD-1679: Weapon System Software Development (Section 5.3, Programming Standards), 1 Dec. 1978.

[46] S. Mittra, Decision Support Systems: Tools and Techniques. New York; John Wiley and Sons, 1986.

[47] D.L. Nazareth, "Issues in the verification of knowledge in rule based systems," International Journal of Man-Machine Studies, vol. 30, pp. 255-271, 1989.

[48] T.A. Nguyen, W.A. Perkins, T.J. Laffey, and D. Pecora, "Knowledge base verification," AI Magazine, vol. 8, pp. 69-75, 1987.

[49] R.M. O'Keefe, O. Balci, and E.P. Smith, "Validating expert system performance," IEEE Expert, vol. 2, pp. 81-90, 1987.

[50] L. Press, "Expert System Benchmarks," IEEE Expert, vol. 4, pp. 37-44, 1989.

[51] R. S. Pressman, Software Engineering: A Practioner's Approach. New York: McGraw-Hill, 1982.

[52] S.L. Riedel, and G.F. Pitz, "Utilization-oriented evaluation of decision support systems," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-16, pp. 980-996, 1986.

[53] A.J. Rockmore, L. Hemphill, R.A. Riemenschneider, M.L. Donnell, K. Gates, Decision Aids for Target Aggregation: Technology Review and Decision Aid Selection. (PAR Report #82-32). New Hartford, NY: PAR Technology Corp.

[54] F.W. Rook and J.W. Croghan, "The knowledge acquisition activity matrix: A systems engineering conceptual framework," IEEE Transactions on Systems, Man, and Cybernetics, in press.

[55] J. Rushby, Quality Measures and Assurance for AI Software. (NASA Contractor Report 4187) Washington, DC: National Aeronautics and Space Administration (Code NTT-4), 1988.

[56] T.L. Saaty, The Analytic Hierarchy Process. New York: McGraw Hill, 1980.

[57] A.P. Sage, "A case for a standard for systems engineering methodology," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC 5, pp. 156-160, 1977.

[58] E.H. Schein, Organizational Psychology. Englewood Cliffs, NJ: Prentice-Hall, 1970.

[59] H.N. Shycon, "All around the model--perspectives on MS applications," Interfaces, vol. 14, pp. 40-43, 1977.

[60] J.R. Slagle and M.R. Wick, "A method for evaluating candidate expert system applications," AI Magazine, vol. 9, pp. 44-53, 1988.

[61] R.A. Stachowitz, C.L. Chang, and J.B. Combs, "Research on validation of knowledge-based systems," Proceedings of the AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems, St. Paul, MN, Aug. 20, 1988.

[62] T.R. Stewart, W.R. Moninger, J. Grassia, R.H. Brady, and F.H. Merrem, Analysis of Expert Judgment and Skill in a Hail Forecasting Experiment. Boulder, CO: Center for Research on Judgment and Policy at the University of Colorado, 1988.

[63] R.M. Tong, N.D. Newman, G. Berg-Cross, and F. Rook, Performance Evaluation of Artificial Intelligence Systems, Mountain View, CA: Advanced Decision Systems, 1987.

[64] J.W. Ulvila, P.E. Lehner, T.A. Bresnick, J.O. Chinnis, and J.D.E. Gumula, Testing and Evaluating C^3I Systems That Employ Artificial Intelligence. Reston, VA: Decision Sciences Consortium, Inc., 1987.

[65] Webster's Seventh New Collegiate Dictionary. Springfield, MA: G & C Merriam Company, 1966.

[66] R. K. Yin, Case Study Research: Design and Methods. Beverly Hills, CA: Sage, 1984.

[67] V.L. Yu, L.M. Fagan, S.M. Wraith, W.J. Clancey, A.C. Scott, J.F. Hanigan, R.L. Blum, B.G. Buchanan, and S.N. Cohen, "Antimicrobial selection by a computer: A blinded evaluation by infectious disease experts," Journal of the American Medical Association, vol. 242, pp. 1279-1282, 1979.

Expert Systems With Applications, Vol. 1, pp. 237–248, 1990
Printed in the USA.

0957-4174/90 $3.00 + .00

Testing Knowledge-Based Systems: The State of the Practice and Suggestions for Improvement

MONICA M. CONSTANTINE AND JACOB W. ULVILA

Decision Science Consortium, Inc., Reston, VA, USA

Abstract—*The field of knowledge-based systems has recently recognized the importance of verification, validation, and testing. This paper presents the results of a survey of the testing practices of knowledge-based systems developers. Common testing strategies are reported and analyzed. Factors affecting testing are discussed. A comprehensive approach to evaluation is described. General conclusions and lessons learned are presented.*

1. INTRODUCTION

RECENTLY, there is an increasing awareness of the importance of verifying, validating, and testing sophisticated software such as knowledge-based systems. In a speech to the Annual Symposium of the International Test and Evaluation Association, Deputy Secretary of Defense Donald Atwood stated that, "we tend to underestimate the amount of testing necessary in software-based weapon systems," and cited evaluation of software as "the fastest growing problem in the Pentagon's test programs" (Struck, 1989).

The Department of Defense is not alone in facing this growing problem. Some recent events have shown that software systems which are not tested thoroughly can subject software developers to possible legal liability. For example, "a computerized therapeutic radiation machine has been blamed in incidents that have led to the deaths of two patients and serious injuries to several others. The deadly medical mystery posed by the machine was finally traced to a software bug, 'Malfunction 54,' named after the message displayed on the operator console" (Joyce, 1987).

Although this example is extreme, it clearly illustrates the important role of software verification, validation, and testing (VV&T). As knowledge-based systems move from research prototypes to operational systems in markets such as nuclear power, aerospace, defense, finance, manufacturing, science, and medicine, more and more managers and developers are realizing the need for more rigorou testing.

Despite the increasing need for more rigorous testing of knowledge-based systems, tools, procedures, and methods have only recently become more prominent in the literature. N.L. Sizemore of Comarco Inc. in Sierra Vista, AZ, maintains an annotated bibliography of "Materials Related to Testing Expert Systems." This bibliography includes journal articles, book chapters, and conference presentations related to verification, validation, and testing of expert systems. As of October 1989, the bibliography contained a total of 135 entries. Over 70% of the entries appeared since 1987, and over 30% of the entries were from two proceedings, the August 1988 AAAI Workshop on Validation and Testing of Knowledge Based Systems, and the IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge-Based Systems.

A majority of the research has focused on one unique aspect of testing knowledge-based systems—the knowledge base itself. One example is EVA, the Expert System Validation Associate, developed by the Lockheed AI Center. EVA is a comprehensive tool to validate any knowledge base written in an expert system shell. EVA's tools include a structure checker, a logic checker, a semantics checker, a completeness checker, a rule refiner, a control checker, a test case generator, an error locator, and a behavior verifier. The tools can be used as an aid in development, quality assurance, and maintenance of a knowledge-based system (Stachowitz, Chang, & Combs, 1988). At the NASA/Johnson Space Center, research is conducted on inference engine validation, development methods, and support tools such as CRSV (Cross-Reference, Style, and Ver-

This work was supported by the U.S. Army Electronic Proving Ground under contract number DAEA18-88-C-0028. The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation. The authors would like to thank Mr. Robert Harder of the U.S. Army Electronic Proving Ground for his helpful comments and support.

Requests for reprints should be sent to Monica M. Constantine, Decision Science Consortium, Inc., 1895 Preston White Drive, Suite 300, Reston, VA 22091.

ification). CRSV, a companion tool for CLIPS, provides the ability to verify a knowledge base. CRSV includes capabilities for cross-referencing relationships, style checking, and dynamic tracing (Culbert & Savely, 1988). The Intelligent Machine Laboratory at Worcester Polytechnic Institute is investigating methods for representing a rule base as an evidence flow graph for use in verification and validation (V&V). Representing a knowledge base as an evidence flow graph aids in detecting unused inputs and subconclusions, unreachable conclusions, and relationships between inputs and outputs (Becker, Green, Duckworth, Bhatnagar, & Pease, 1989). Bellman and Walter of the Computer Science Laboratory at The Aerospace Corporation have developed an approach to V&V that stresses that knowledge bases are models, and methods to test knowledge bases should both identify and correct errors. The methods include incidence matrices, CART for classifying cases, and techniques to obtain test cases (Bellman & Walter, 1988). Also at The Aerospace Corporation, research is being conducted for establishing criteria that define a rule base as a formal mathematical structure and on algorithms that check the rule base against the specified criteria (Landauer, 1989).

The recent literature stresses the complexity of the knowledge base, the unexpected side effects caused by interactions of rules, and the importance of rigorously testing the knowledge base. However, in practice, these aspects of testing knowledge-based systems are rarely emphasized. For example, Harmon, Maus, and Morrissey (1988) state, "one appealing feature of an expert system is that its knowledge base is built incrementally. Because of this, the knowledge base itself doesn't need a formal test phase."

Although the importance of verification, validation, and testing of knowledge-based systems is gaining prominence in the literature, rigorous VV&T methods have not yet been put into practice. Currently, VV&T presents a difficult challenge for the expert system manager, developer, tester, and user in both the commercial and government markets. This paper presents the results of a survey of testing practices, reports and analyzes current testing strategies, and discusses factors which affect testing. Additionally, a comprehensive approach to evaluation is described and general conclusions and lessons learned are presented.

2. THE STATE OF THE PRACTICE

During 1989, we conducted interviews with 30 members or employees of the U.S. Army who had experience in developing and testing knowledge-based systems. The individuals represented seven different organizations in the Army: Department of the Army, Army Materiel Command, Army Research Institute, Defense Logistics Agency, Training and Doctrine Command, Forces Command, and Health Services Command.

The interviews, performed in person or over the telephone, were relatively open-ended. Generally, the interviewees were asked to: provide a brief description of the system, indicate why the system was considered artificial intelligence (AI), provide information on how the requirements for the system were specified, describe the development environment, report on the use of experts (single or multiple) in developing or testing the system, state how the system was tested, and specify what was perceived to be the greatest difficulty or biggest stumbling block in development and testing.

2.1. Knowledge-Based Systems in the Army

At the time of the survey, knowledge-based systems in the Army were moving from the research labs and into the organizational areas. There were a large number of prototype systems in the various organizations within the U.S. Army; some were operational, few had been fielded, and even fewer were building up a track record. The large number of prototypes is not unusual. For example, there are over 2000 expert systems in Japanese industries and many of them remain as prototypes (Terano & Kobayashi, 1989). The large number of prototypes may be indicative of the relatively new position of knowledge-based systems outside the research laboratories.

Movement of AI techniques from the research departments was illustrated by the establishment of AI centers. Organizations within the Army were establishing working groups committed to using AI techniques to solve problems, improve productivity, and lower costs. Organizations, such as the Defense Logistics Agency and the Electronic Proving Ground, were also providing in-house training on selecting valid AI projects, use of shells, and knowledge engineering. The AI centers usually mixed a number of small, short-term AI projects with one large, longer-term project. The success of a few small projects often paved the way for a commitment from management to devote the time and resources for a more costly, longer-term project.

Knowledge-based system projects in the Army covered a diverse range of applications, many of which were similar to commercial applications. Knowledge-based systems had been applied to many different tasks, including hazardous material classification, selection of appropriate contract clauses, in-house support for medical research (human vaccine testing and immunization scheduling), assignment of ratings for psychiatric disability compensation, battle management, and diagnostics for helicopter repair. Many of the knowledge-based systems used information from a single or multiple experts in a domain area and incorporated information into the knowledge bases from official doctrine or regulations. The systems ranged from a number of small "in-house" aids that contained about

50 rules to a very large system that had over 400 procedures, each containing its own set of rules. The systems were designed to run on a variety of machines—personal computers, main frames, Sun workstations, or Symbolics.

The systems were designed for use by both experts and nonexperts. Typically both experts and nonexperts interfaced with the system in a similar manner (answering a series of questions or making selections from a menu), but the expert was able to use his or her knowledge of the problem area, while the nonexpert had to use other sources such as invoices, billing statements, or patient summaries. The difference in end users suggests that different aspects of the system should be emphasized, depending on the end user. For example, more emphasis may be needed on the interface, stress testing, and the quality of the explanations when a system is designed for a nonexpert rather than an expert, but such use-dependent testing was rarely done.

Despite the diversity, the systems were all quite similar in their goals and in the methodology used in development. Most, but not all, of the systems performed a support function where high dollar costs or loss of life were not involved. Generally, the systems functioned as aids to a person making a decision, rather than as decision maker. Many of the systems were not designed to outperform an expert, but to add consistency to a process routinely performed by many individuals within an organization. Few expert systems were designed as a tutorial for a nonexpert. One of the systems designed to train new employees in an administrative task seemed to have limited capabilities. Rather than teaching the nonexpert the skills necessary to become an expert, a user commented that the system gave "1 year of experience 20 times."

Most of the systems were developed through rapid prototyping. For the most part, requirements documents did not exist. Rather, the software either evolved from an existing software program or was created informally to meet a need of a particular individual or department within the organizations. Typically the systems were developed using expert system shells (EXSYS, M-1, CLIPS, 1st Class, Level5, ART) or in LISP, which many developers saw as a rapid prototyping language. When a system became a "final product," some developers saw a need to convert it to another language, such as Ada or C. This conversion is a potential problem—one that could be very expensive and even infeasible. Furthermore, the conversion has implications for testing, since much of the testing performed on the original system will need to be repeated on the converted system.

As stated earlier, most of these systems developed as prototypes. However, in some cases there were plans to convert the prototype to an operational system. The requirements documents for these systems generally seemed to focus on system integration issues—such as access to large databases or converting to a different shell, a different development language, or a different hardware configuration—and not on issues relating to the expected performance of the software. The lack of documented requirements and the "fuzzy" nature of a knowledge-based system add to the difficulty of testing it.

2.2. Testing Knowledge-Based Systems

In asking what it took to test a knowledge-based system, we received a range of responses. Some respondents thought that testing AI software was not any different from testing conventional software ("a rule base is just a structured database"); others believed that testing AI software was very different and required a multifaceted approach, to include static testing, dynamic testing, multiattribute analysis, acceptance by experts, and questionnaires to users.

Many of the difficulties faced in testing knowledge-based systems can be generalized:

- *Issues concerning testing were not raised early enough in the development process.* Not only is it important to detect errors early, it is also important to start planning how the system will be tested. According to Bellman and Walter (1989), "preparing for testing begins with system design." Building a testable knowledge-based system is building a model and requires specifying acceptable performance ranges upfront and following a sound development methodology. "Rapid prototyping is not an excuse for poor plans and task analysis, quick fix or kludges" (Bellman & Walter, 1989).
- *In a resource-constrained environment, it was difficult to test thoroughly.* The experiences of Science Applications International Corporation indicate that V&V and testing can consume as much as half the project dollars and is usually one of the first areas to get cut when budgets get tight (Miller, 1989). Budget constraints may necessitate cutting back or completely eliminating desirable software tests. For example, it may not be feasible to bring in an expert who was not involved in developing the knowledge-based system to help test it (Bachant, 1988).
- *Often what the knowledge-based system should do was not clearly defined or made explicit, which added to the difficulty of testing.* The importance of documenting requirements is clearly stated in most of the literature, yet documented requirements rarely exist. Culbert and Savely (1988) contend that "without a clear definition of what the system is supposed to do, it is literally impossible to understand how to test it." Naser (1988), of the Electric Power Research Institute, states that "V&V is hampered by the lack of stable documentation" and that a requirements document should "provide external performance goals that can be confirmed by tests." Further re-

TABLE 1
Testing Strategies

Testing Strategy	Description	Charactenstics of Development Environment
PROTOTYPE FOREVER	The expert receives the latest version of the software and uses it in an actual work setting. The expert monitors the system in use and provides feedback on the interface, the explanation facility, and the reasonableness of the system's outputs over time.	• Informal development environment • No documented requirements • Developed for in-house use by a particular expert • Developer and expert user have a close working relationship • System provides a support function where cost of error is low
AGREEMENT (The ultimate standard is agreement between the system and a panel of experts on a set of test cases)	As the system is being developed, it is tested with the expert. When an initial version of the system is complete, a sample of test cases is selected based on actual data. The test cases are given to a panel of experts who are asked to determine the outcome. The same set of test cases is presented to the system and the system determines the outcome. The system passes the "test" if the system and panel of experts agree on the outcome for some percentage (85%, for example) of the test cases. The system is put into use and monitored over time.	• Informal development environment • No documented requirements • System developed for in-house use • Expert is not always the intended user • System serves as an aid to a decision maker rather than as a decision maker
COMPLIANCE (The ultimate standard is compliance with a format specified by regulations)	Test cases selected based on actual data from a previous year where the outcome is known. Those cases are presented to the system and the appropriate changes are made. Another set of test cases is selected from current data where the outcome is not yet known. These cases are presented to the system and the output is correct if it complies with the relevant regulations.	• Informal development environment • No documented requirements • System designed to meet an in-house need • Rules in the knowledge base are based on regulations • System performs a support function and is designed to add consistency to an existing inconsistent process
SATISFACTION (The ultimate standard is the satisfaction expressed by the end user and/or expert)	The developer examines the knowledge base for missing rules, rules that can be collapsed, and rules that are not being fired. The expert assesses the correctness of the rules, the quality of the explanations, and the quality of the answers. The user assesses his or her ability to interface with the system, the timeliness of the response, the reasonableness of the outputs and explanations, and how the system fits in with the operating environment.	• Relatively informal development environment • No documented requirements • Developer and user have a relatively distant relationship • System designed to function as an aid to the decision maker
CASE-DEPENDENT (The final assessment of how well the system is doing is dependent on agreement with the expert using a sample of test cases that were also used in development)	The developer examines the knowledge base, assesses the effect of adding rules, determines if rules can be combined, and looks for errors. A large sample of test cases is selected that approximate the population of cases the system will receive. The expert assesses the answer to test cases without using the system. Then, the expert uses the system to obtain outputs (all of the expert's actions are saved). The saved data reflecting the expert's actions are analyzed and changes are made to the knowledge base. It is	• No documented requirements • Relatively informal development environment • Expert is available for development and testing • The system is designed primarily to add consistency to an existing process

TABLE 1 (ctd.)

Testing Strategy	Description	Characteristics of Development Environment
	necessary for the expert and system to agree 80% of the time. The system is then tested with the intended users (who are not experts). The nonexperts interpret the input data from summary sheets and the differences in data input between the expert and nonexpert are examined and appropriate changes are made to the system.	
ORGANIZATIONAL TESTING (The ultimate goal is to have the system—a training system—improve performance in an organizational setting)	The interface is iteratively evaluated by ultimate end user. Interface evaluation includes an assessment of screen design, feedback message placement, scrolling, features, menu naming, design and actions. The system is evaluated in a classroom setting by observing the system in use and administering questionnaires. Observers videotape and take notes to assess how both students and instructors use the system in an actual classroom setting. Questionnaires are administered to both students and instructors to gather information regarding features used, perceived usefulness, perceived problems, and general feelings. An experiment, using subjects in an actual classroom environment, is designed to assess the effect of using the system on student performance.	• Research-oriented development environment • Careful, planned development • Sophisticated system designed as an intelligent training aid in a classroom setting
FIELD TESTING (The ultimate standard is to assess whether the system actually reduces time and cost of the existing process in the field)	Each prototype is tested with past cases from saved actual data. Test in a similar operational environment for 3 months and obtain feedback on system effectiveness and user interface. Appropriate changes are made to the system. The system is then run in parallel to the existing process in the intended operational environment for approximately one year. During the parallel test, assessments are made as to how well the system is meeting the goals stated in the requirements document.	• Requirements document • System developed by outside contractor • More formal development environment • System designed to reduce time and cost of an existing process
MULTI-FACETED (The ultimate goal is to test all aspects of the system)	Developer performs comprehensive static analysis of the knowledge base using automated tools. Dynamic testing is performed to test the system with an expert using a comprehensive set of test cases not used in development. Multiattribute analysis is used to obtain subjective measures for system performance. System is tested with "developer" experts as well as outside experts. Questionnaires are administered to both developer experts and outside experts.	• Formal development environment • Documented requirements • Documented test plan • Specially developed tools for static analysis

search sponsored by the Electric Power Research Institute indicates that there may be as many as five different groups interested in the system's overall quality—the development team, the sponsors, the users, the experts used in development, and the market experts or critics. The different groups may all have different interests and different performance expectations. The requirements document should provide the criteria used in the V&V process and serve as an agreement between the different groups by explicitly stating the performance requirements (Kirk & Murray, 1988). Rushby (1988) suggests that, while it may be too difficult to explicitly define what the system should do up front, it may be entirely

possible to "specify certain undesired or safety properties quite sharply." This suggests that the requirements document should define the "minimum competency requirements" and tests should be performed to ensure, that at the very least, the system conforms to these criteria.

- *The complexity of the knowledge base and the lack of tools for static analysis were problems.* Although static analysis tools are discussed in detail in the literature, few testers have access to such tools. Most shells do not provide extensive utilities for static analysis and most developers do not have sufficient time or resources for an extensive manual analysis of the knowledge base. One exception is CRSV (Cross Reference Style and Verification Tool) which has recently been made available and is distributed with CLIPS.
- *The unavailability of the expert for testing was a problem.* Many respondents commented that, although they expected the experts to be available to test the software, they were not. Generally, if the experts were unable to test the system, the project failed.

2.3. Determinants of Testing

Generally, the level of testing performed on knowledge-based systems seemed to depend on four factors: information available on testing methods and procedures, time constraints, resource constraints, and characteristics of the development environment (such as formality and accessibility of the developers to the users). Typically, when the developer and user had a close working relationship, the system was put into use and the developer worked with the user to implement the required software changes on an "as-needed" basis.

Some of the strategies used to test knowledge-based systems along with corresponding characteristics of the development environment are listed in Table 1. At some level, all of the testing strategies addressed correctness of the answer and correctness of the reasoning. All of the systems were judged against some standard, although in many cases that standard was simply a vague notion of "correctness." For example, in the agreement-testing strategy, the system's output was stamped "correct" *if it agreed with the outcomes* specified by a panel of experts for 85% of the test cases. In case-dependent testing, great care was taken to select a set of test cases that approximated the kinds of cases the system would be expected to handle in an operational environment, but all of the cases used in testing were also used in development. With the field testing strategy, the system was judged "correct" if it met a few performance criteria (such as a 30% reduction in downtime) specified in a requirements document.

In one particular project—an intelligent tutoring system—testing attributes such as correctness of the outcome was not as important as testing other attributes such as the correctness and clarity of the reasoning, the usability of the system, and how well the system fit into the intended operational environment. Testing included using the system in an actual classroom setting, videotaping the system in use, and taking extensive notes while observing the system in use. Both the students and instructors were administered questionnaires designed to assess features used, perceived usefulness, perceived problems, and general feelings about the system. Additionally, an experiment using a control group and a test group of students was conducted to determine whether or not the system actually improved student performance. Generally, the other testing strategies did not sufficiently address either usability or fit with the organization.

Many of the testing strategies (prototype forever, agreement, compliance) did not directly address either the structure or content of the knowledge base. The satisfaction and case-dependent strategies insufficiently addressed testing the knowledge base due to the lack of automated tools for static analysis. Most of the developers realized the importance of structural or static testing, but without automated tools, lacked the resources or time to do as much testing as they would have liked. Only the multifaceted testing strategy used an automated static analysis tool especially adapted for the particular application.

Most of the knowledge-based systems made use of one of the many expert system shells available on the market, and no testing was aimed specifically at the inference engine of the shell. Most developers assumed that, when they purchased a shell, the inference engine had already been tested thoroughly. This may not be the case. The literature indicates that perhaps only one inference engine, CLIPS, has been formally validated (Culbert & Savely, 1988).

The testing strategy of changing inputs, obtaining outputs, and asking the expert if the results are reasonable may be appropriate for small, expendable, nonautonomous, noncritical, in-house systems (where the cost of an error and the cost of the system are extremely low), in environments where the developer and the expert work closely together and the performance of the system is continually monitored. But be prepared: this system is likely to remain a prototype forever. When a knowledge-based system is to be used by a large number of individuals, to replace an existing method for solving a particular problem, or to perform an important or critical function or where the cost of a system error may be high, more rigorous and thorough testing is necessary. One approach for more comprehensive test and evaluation may be the multiattribute framework recommended by Adelman and Ulvila (in press).

2.4. MAU Approach to Evaluating a Knowledge-Based System

Adelman and Ulvila (in press) describe an evaluation hierarchy that encompasses the following kinds of tests:

- *Service Tests.* Service tests address the design and portability of the computer system, computer usage, and system integration issues.
- *Structural Tests of the Knowledge Base.* Static or structural testing is concerned with examining the underlying structure of the knowledge base, that is, the logical consistency and logical and functional completeness of the rules. Tests for logical consistency are aimed at finding and correcting redundant rules, subsumed rules, conflicting rules, and unnecessary if conditions. Tests for logical completeness are for finding unreferenced attribute values, illegal attribute values, unreachable conclusions, and dead ends in the knowledge base. Functional completeness measures the extent to which the knowledge base addresses all the domain problems that the users have to or need to address.
- *Content-Specific Tests for the Knowledge Base.* In this type of testing, the domain expert is asked to make judgments about the accuracy and adequacy of the embedded knowledge. The judgments may be elicited individually with single experts or using group techniques with multiple experts. In this category of tests, the domain expert is asked to assess the accuracy of the facts in the knowledge base, the accuracy of the embedded rules, the acceptability of the knowledge representation scheme, the adequacy of the source, and the modifiability of the knowledge base.
- *Inference Engine.* Specific tests are aimed at determining the correctness of the inference engine.
- *Performance Tests.* Performance tests are aimed at determining how well the system carries out its designated functions. These tests can be divided into functions for which "ground truth" answers exist and those for which no ground truth exists. Where ground truth exists, the performance of the system can be compared with a known standard. Where no ground truth exists, we must rely on the judgment of experts to assess the quality of the conclusion. Performance measures should also include measures of response time and time to accomplish the task.
- *Usability Tests.* Usability measures incorporate a number of factors that relate to how well the computer is adapted to the needs of the user. Usability measures can be assessed by observation or opinion survey. Measures of usability that can be observed are the extent of use, manner of use, and features used. Usability measures assessed by opinion survey include confidence in the system, ease of use, acceptability of the interface, acceptability of the results, scope of the application, adequacy and clarity of the explanations, and impact on the organization.

The characteristics or attributes described above were

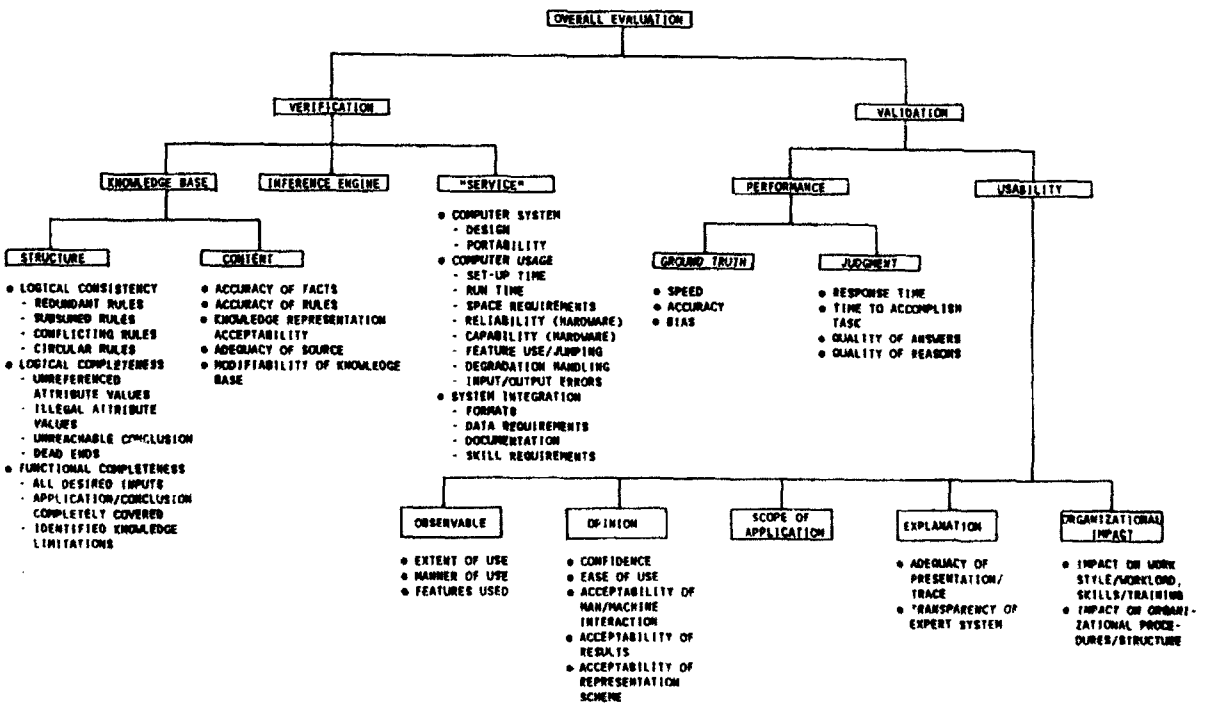


FIGURE 1. A MAUA Framework for integrating test and evaluation criteria.

TABLE 2
Testing Attributes and Definitions

Higher-Level Attribute	Attribute	Definition
KB STRUCTURE		
Logical Consistency	Redundant Rules	Individual rules or groups of rules that essentially have the same conditions and conclusions. Redundancy can be due to duplicate rules or the creation of equivalent rules (rule groups) by wording variations in the names given to variables, or the order in which they are processed.
	Subsumed Rules	When one rule's (or group of rules') meaning is already expressed in another rule (or group of rules) that reaches the same conclusion from similar but less restrictive conditions.
	Conflicting Rules	Rules (or groups of rules) that use the same (or very similar) conditions, but result in different conclusions, or rules whose combination violates principles of logic (e.g., transitivity).
	Circular Rules	Rules that lead one back to an initial (or intermediate) condition(s) instead of a conclusion.
	Unnecessary If Conditions	The value on a condition does not affect the conclusions of any rule.
Logical Completeness	Unreferenced Attribute Values	Values on a condition that are not defined; consequently their occurrence cannot result in a conclusion.
	Illegal Attribute Values	Values on a condition that are outside the acceptable set or range of values for that condition.
	Unreachable Conclusion	A conclusion that cannot be triggered by the rules combining conditions.
	Dead Ends	Rules that do not connect input conditions with output conclusions.
Functional Completeness	All Desired Inputs	The knowledge base can handle all input conditions that need to be addressed.
	Application/Conclusion Completely Covered	The knowledge base can trigger all output conclusions that need to be addressed.
	Identified Knowledge Limitations	The rules in the knowledge base can tell the user if input conditions currently being processed cannot be addressed. Analogously, if the expert system is such that a user can specify a conclusion in order to identify the input conditions that would generate it, an expert system that was knowledgeable of its limitations would tell users if a conclusion currently being processed as input could not be addressed.
KB CONTENT		
Accuracy & Adequacy	Accuracy of Facts	The quality of the unconditional statements in the knowledge base.
	Accuracy of Rules	The quality of the conditional statements in the knowledge base representing expert judgment.
	Knowledge Representation Acceptability	Whether or not the scheme for representing knowledge is acceptable to other (a) domain experts and (b) knowledge engineers.
	Adequacy of Source	Quality of the persons or documentation used to create the knowledge base.
	Modifiability of Knowledge Base	The extent to which the knowledge base can be changed.
	—*Control Over*	The extent to which changes in the knowledge base are limited to certain classes of users.
	—*Expandability* (by human/machine)	The extent to which the knowledge base can be increased (or decreased) by users (or their representatives) or by the knowledge base (i.e., the machine) itself through learning.
SERVICE		
Computer System	Design	The extent to which the expert system runs on the approved computer hardware and operating system and utilizes the preferred complement of equipment and features. In some cases, the original system requirements may specify or describe the preferred or required system. In other cases, the tester may need to survey available equipment at the intended installation.
	Portability	The ease with which the expert system can be transferred to other computer systems.

TABLE 1 (cont.)

Higher-Level Attribute	Attribute	Definition
Computer Usage	Set-Up Time	The amount of time required for the computer operator to locate and load the program (if any) and the time to activate the program. Set-up time should be measured in the expected operating environment (i.e., how the program will actually be implemented).
	Run Time	The amount of time required to run the program with a realistic set of input data. This factor refers only to the time that the computer program takes to run; the time needed for the programmer and user is included under dynamic testing factors
	Space Requirements	The amount of RAM and disk space required by the program.
	Reliability (Hardware)	Percentage of time the computer system could be expected to be operating effectively.
	Capability (Hardware)	The computer system's total amount of RAM and disk space.
	Effect of Feature Use/ Jumping	The extent to which moving from various parts of the program causes errors.
	Degradation (Graceful?)	How well the program (a) saves data and analyses, and (b) permits continuation after an unexpected program or system crash or power outage.
	Handling Input Errors	The extent to which the program (a) prohibits a program crash, and (b) tells the user what to do after an input mistake.
System Integration	Formats	The extent to which the program uses input and output formats that are consistent with the intended use. This includes any mandated or standard formats that are specific to the intended user organization.
	Data Requirements	The extent to which the program's data requirements are consistent in content, quantity, quality, and timeliness with those available to the intended user organization. The program should also be able to interact with specified and appropriate databases and communications systems.
	Documentation	The adequacy of material regarding the program's use and maintenance. User's manuals should be complete and understandable. Copies of computer code and its supporting documentation should be complete and understandable, and should allow maintenance by the organization. (All applicable software documentation standards should be met.)
	Skill Requirements	The extent to which the program can be operated by appropriately skilled individuals. The appropriate skill requirement includes job description, users' technical background, and training requirements. The appropriate level may be specified in requirements or may be determined by reference to the organizational setting of its intended use and to the personnel assigned to that setting.
PERFORMANCE		
Ground Truth	Speed	The amount of time it takes a user working with the expert system to solve representative problem scenarios.
	Accuracy (d')	The degree of overlap in the distributions of belief values when the hypothesis is true versus false (see Lehner, 1989).
	Bias (B")	The difference in the proportion of false negatives (hypothesis is true but system says false) to false positives (hypothesis is false, but system says it's true) (see Lehner, 1989).
Judgment	Speed: Response Time	The judgments of users regarding the adequacy of the amount of time the expert system takes to react to user inputs.
	Speed: Time to Accomplish Task	The judgments of users regarding the adequacy of the amount of time required to perform the task when using the expert system.
	Quality: Quality of Answers	The judgments of users and experts regarding the system's capability.
	Quality: Quality of Reasons	The judgments of users and experts regarding the adequacy of the system's justification for its answers.
USABILITY		
Observable	Extent of Use	How much users employ the expert system to perform the task (e.g., the proportion of the time that the system was used for task accomplishment)

TABLE 2 (ctd.)

Higher-Level Attribute	Attribute	Definition
	Manner of Use	The way in which users employ the system and its features, including (1) the procedures to access different modules, (2) the way that intermediate and final outputs are incorporated into the user's results, and (3) the use of man–machine interfaces.
	Features Used	The extent to which different aspects of the expert system are employed by users.
Opinions	Confidence	How confident users feel in taking actions based on working with the expert system.
	Ease of Use	How easy users judge the system is to use after they have completed training and become familiar with the system.
	Acceptability of the Person/Machine Interaction Process	The extent to which users assess that they and the system are performing the tasks/activities for which they are best suited.
	Acceptability of the Results	The users' judgments regarding the adequacy of the system's capability.
	Acceptability of Representation Scheme	The users' judgments regarding the adequacy of the system's way of presenting knowledge.
Scope	Scope of Application	The users' judgments regarding the adequacy of the expert system in addressing domain problems.
Explanations	Adequacy of Presentation/ Trace	The users' judgments regarding the acceptability of the system's presentation of its reasoning process.
	Transparency of Expert System	The extent to which the system's reasoning process is clear and understandable to its users.
Organizational Impact	Impact on Work Style/ Workload, Skills & Training	The judgments of users regarding the impact of the expert system on (a) how they do their job, or (b) the skills and training required to perform it effectively.
	Impact on Organizational Procedures & Structure	The judgments of users regarding the impact of the expert system on the organization's operations.
	Input–Output	The users' judgments regarding the adequacy of all the expert system's displays except those tracing the reasoning process.

put into a multiattribute utility (MAU) model for evaluating a knowledge-based system (Ulvila, Lehner, Bresnick, Chinnis, & Gumula, 1987). Multiattribute analysis provides an overall framework for evaluation of a system where multiple objectives are important. MAU provides a formal structure for conceptualizing measures of effectiveness (attributes), a mechanism for decomposing global measures into component dimensions and for reintegrating them to yield a summary measure of value. Adelman and Donnell (1986) present a case study where multiattribute analysis was successfully used to evaluate an expert system prototype. In the study, Adelman and Donnell conclude that the MAU framework evaluated the expert system rigorously and allowed for empirical data to be collected that could be used to improve the design, development, and implementation of decision support software.

When applying MAU to the evaluation of an expert system, the system is conceptually decomposed into attributes that can be defined well enough so that measures of how well the expert system performs on each attribute can be obtained. The attributes can be assigned different weights to assess their relative importance to each other as well as their relative importance at the various stages in the system's life cycle. The MAU evaluation hierarchy is displayed in Figure 1. The evaluation hierarchy encompasses both verification and validation. In this MAU framework verification includes all attributes falling under "Knowledge Base," "Service," and "Inference Engine," while validation includes "Performance" and "Usability." The definitions for each attribute are presented in Table 2.

3. LESSONS LEARNED

Most of the lessons from the U.S. Army's experience in testing knowledge-based systems are applicable to all expert systems development. Generally, the lessons relate to requirements and issues that were not raised early enough in the development process. Emphasis was generally placed on developing a product that could be seen, touched, and operated. As a consequence, documentation and testing were often "crowded out." The list below summarizes some of the lessons learned and makes some suggestions for developing testable knowledge-based systems.

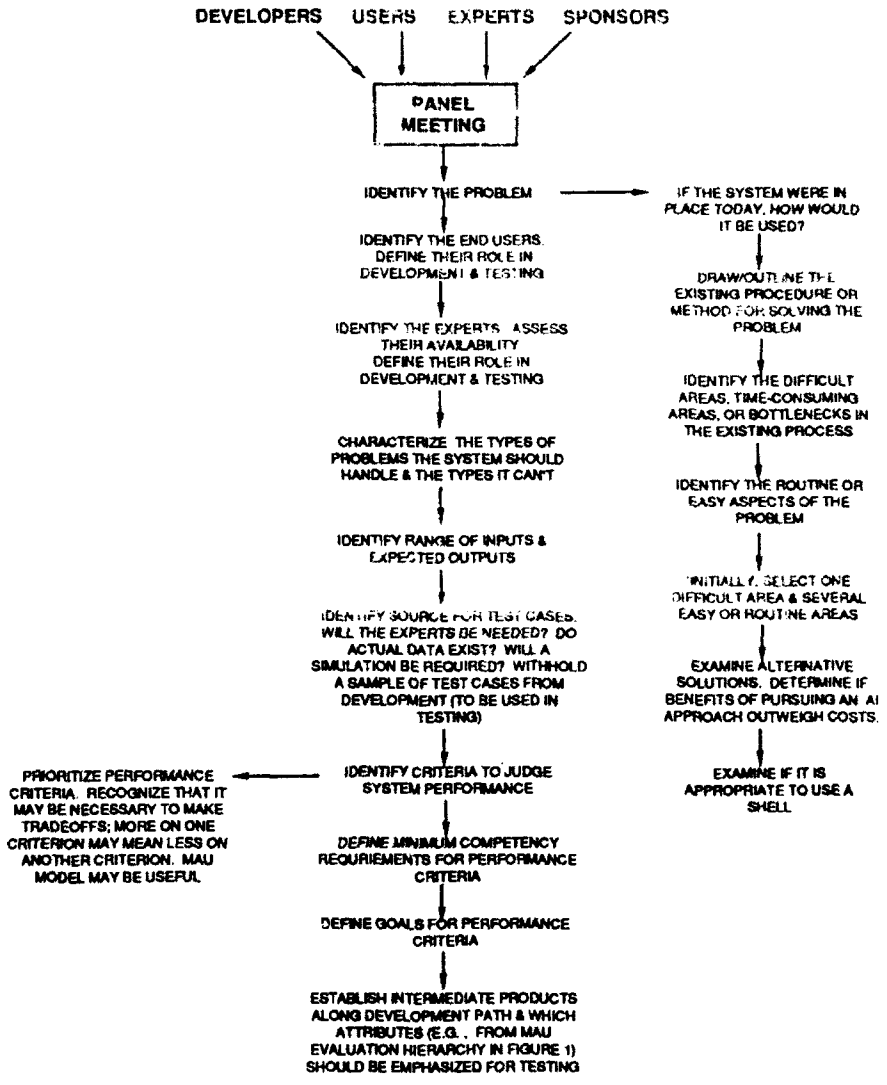


FIGURE 2. Requirements generation process.

1. *Specify system requirements.* It is impossible to test the system without a clear understanding of what the system should do. If it is too difficult to do up front, do it later. Develop a prototype and recognize it for what it is—a means to generate requirements. Figure 2 displays an outline for a possible requirements generation process.
2. *Give the knowledge-based system an apprenticeship period in the intended operating environment.* Keep a library of cases presented to the system, observe the system in use, and monitor its performance. An apprenticeship period may be essential for assessing how the system will perform in an operational environment and for finding errors not found in previous testing efforts.
3. *Instill order.* Establish a set of products (tools, shells) to use, provide training, and build your own tools to support testing if the vendor will not provide them. Automated tools for static analysis are essential for thoroughly testing a knowledge base and should be available to the developers of knowledge-based systems. Since there are no industry standards for knowledge-based systems, specify formats, naming conventions, procedures for commenting rules, and other programming standards for developers in a particular organization to use when building knowledge bases and knowledge-based systems.

To date, successful applications seem to be limited in scope and solve relatively well-defined problems. Many of those interviewed seemed to think that the future of AI is in "little modules of AI" as part of larger systems, rather than in large AI systems. Generally, prototypes are regarded as useful and valuable for "studying the problem" but do not necessarily lead to operational systems. Prototyping may be nothing more than one step on the way to defining system requirements. To develop operational, usable, and testable systems,

developers need to work harder at answering the difficult questions up front, to define explicitly the scope of the application, the input domain, the limitations of the system, the expected level of performance, the usability requirements, and to rigorously test those aspects of the system. Without rigorous testing, a system cannot be a reliable and useful contributor.

REFERENCES

Adelman, L., & Donnell, M.L. (1986). Evaluating decision support systems: A general framework and case study. In S. Andriole (Ed.), *Microcomputer decision support system*. Wellesley, MA: QED Information Sciences, Inc.

Adelman, L., & Ulvila, J.W. (in press). Evaluating expert system technology. In S. Halpin & S. Andriole (Eds.), *Information technology for command and control*. New York: IEEE Press.

Bachant, J. (1988, August). *Validating and testing XCON*. Paper presented at the First AAAI Workshop on Validation and Testing of Knowledge-Based Systems, Palo Alto, CA.

Becker, L.A., Green, P.E., Duckworth, J., Bhatnagar, J., & Pease, A. (1989, August). *Evidence flow graphs for VV&T*. Paper presented at IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge-Based Systems, Detroit, MI.

Bellman, K.L., & Walter, D.O. (1988, August). *Analyzing and correcting knowledge-based systems requires explicit models*. Paper presented at the First AAAI Workshop on Validation and Testing of Knowledge-Based Systems, Palo Alto, CA.

Bellman, K.L., & Walter, D.O. (1989, April). *Designing knowledge-based systems for reliability/performance*. Notes for a one-day tutorial course. Presented at The Conference on Testing Computer Software, Washington, DC.

Culbert, C., & Savely, R. (1988, August). *Expert system verification and validation*. Paper presented at the First AAAI Workshop on Validation and Testing of Knowledge-Based Systems, Palo Alto, CA.

Harmon, P., Maus, R., & Morrissey, W. (1988). *Expert system tools and applications*. New York: John Wiley & Sons, Inc.

Joyce, E. (1987). Software bugs: A matter of life and liability. *Datamation*, 33, 88–92.

Kirk, D.B., & Murray, A.E. (1988). *Verification and validation of expert systems for nuclear power applications* (Report). Palo Alto, CA: Electric Power Research Institute.

Landauer, C. (1989, August). *Principles of rule base correctness*. Paper presented at the IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge-Based Systems, Detroit, MI.

Lehner, P.E. (1989). Toward an empirical approach to evaluating the knowledge base of an expert system. *IEEE Transactions on Systems, Man, and Cybernetics*, 19, 658–662.

Miller, L.A. (1989, August). *A comprehensive approach to the verification and validation of knowledge-based systems*. Paper presented at the Workshop on Verification, Validation and Testing of Knowledge-Based Systems, IJCAI Conference on Artificial Intelligence, Detroit, MI.

Naser, J.A. (1988, August). *Nuclear power plant expert system verification and validation*. Paper presented at the First AAAI Workshop on Validation and Testing of Knowledge-Based Systems, Palo Alto, CA.

Rushby, J. (1988, August). *Validation and testing of knowledge-based systems—How bad can it get?* Paper presented at the First AAAI Workshop on Validation and Testing of Knowledge-Based Systems, Palo Alto, CA.

Stachowitz, R.A., Chang, C.L., & Combs, J.B. (1988, August). *Research on validation of knowledge-based systems*. Paper presented at the First AAAI Workshop on Validation and Testing of Knowledge-Based Systems, Palo Alto, CA.

Struck, M. (1989). Evaluation of weapon software critical problem for DoD, Atwood says. *Defense News*, October 2, 20.

Terano, T., & Kobayashi, S. (1989, August). *Problem analyses, tool evaluation, and verification & validation*. Paper presented at the IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge-Based Systems, Detroit, MI.

Ulvila, J.W., Lehner, P.E., Bresnick, T.A., Chinnis, J.O., Jr., & Gumula, J.D.E. (1987). *Testing and evaluating C^3I systems that employ artificial intelligence* (Technical Report 87-7). Reston, VA: Decision Science Consortium, Inc.

A NOTE ON THE APPLICATION OF CLASSICAL STATISTICS TO EVALUATING THE KNOWLEDGE BASE OF AN EXPERT SYSTEM[1]*

Paul E. Lehner

Information Systems and Systems Engineering
George Mason University
4400 University Drive
Fairfax, VA. 22030

and

Decision Science Consortium, Inc.
1895 Preston White Drive
Reston, VA. 22090

Jacob W. Ulvila

Decision Science Consortium, Inc.
1895 Preston White Drive
Reston, VA. 22090

ABSTRACT

This paper explores the use of classical statistical procedures for evaluating the knowledge base of an expert system that outputs quantitative belief values. Statistical procedures can be used to estimate the relative frequency that an expert system will output belief values that (a) strongly support an incorrect hypothesis, and (b) do not provide strong support for any hypothesis. These procedures can also be used to estimate the minimum number of test problems required to evaluate an expert system.

[1]This research was supported, in part, by the U.S. Army Electronic Proving Ground under contract number DAEA18-88-C-0028. The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

*Submitted to IEEE Transactions on Systems, Man, and Cybertics.

A NOTE ON THE APPLICATION OF CLASSICAL STATISTICS TO EVALUATING THE KNOWLEDGE BASE OF AN EXPERT SYSTEM

Paul E. Lehner and Jacob W. Ulvila

1.0 INTRODUCTION

In an earlier paper (Lehner, 1989), an empirical approach was presented to evaluating the knowledge base of an expert system that outputs quantitative belief values (e.g., probability, Shaferian belief, fuzzy membership, certainty value, etc.). That paper emphasized the use of a nonparametric statistical procedures to characterize the extent to which each node in an inference network (or other knowledge structure) distinguishes between true and false instances of each hypothesis tested at that node. This paper extends the discussion in Lehner (1989) by demonstrating how parametric procedures from classical statistics can be used in the same way. The justification for for using these procedures is also discussed.

1.1 The Signal Detection Analogy

As noted in Lehner (1989) an expert system that evaluates predefined hypotheses is loosely analagous to a signal detector. A signal detector is any system that functions to discriminate occurrences from nonoccurrences of a signal. As shown in Figure 1, the signal detection problem is often characterized as one of receiving a set of sample values (perceived signal strength) from one of two distributions (signal exists vs. signal does not exist), and on the basis of this information deciding from which of the two distributions the signals were drawn. Usually this decision is based on whether the observed signal strengths exceed a threshold. The decision threshold is determined from background knowledge of the underlying distributions. The sensitivity of a signal detector is often measured in terms of the normalized difference between the means of the two distributions (d'). If d' is large (small), then the error rate of signal/no signal decisions will be small (large).

In an expert system inference network (or other knowledge structure) each node represents two or more mutually exclusive hypotheses. Most expert systems generate a belief value

$$d' = \frac{\mu_1 - \mu_2}{\sigma}$$

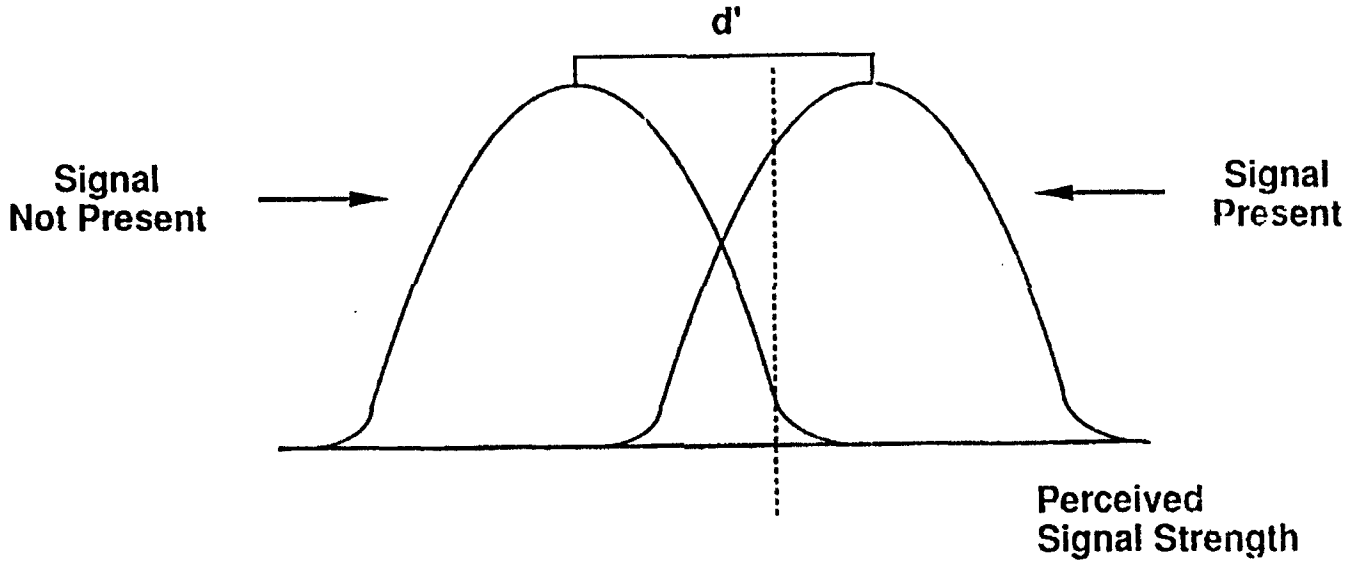


Figure 1: Hypothetical Distribution of Perceived Signal Strength in Signal Detection Theory

(probability, certainty level, Shaferian belief, etc.) for each hypothesis. Consider a node that discriminates two hypotheses, H_1 and H_2. When H_1 is true, we would generally expect the belief value in H_1, bel(H_1), to be higher than when H_2 is true. The user's problem is to use the belief values output by the expert system, along with other available information, to select a hypothesis and act accordingly. If there is a large (small) difference between the mean bel(H_1) value when H_1 vs. H_2 is true, then the expert system should be useful (useless) in helping a user to discriminate these two hypotheses.

1.2 Measures of the Usefulness of an Expert System

From a users perspective, an expert system is useful if it helps discriminate instances when different hypotheses are true. One approach to evaluating a system is to estimate the proportion of times the expert system will generate advice that is useful in discriminating among alternative hypotheses. In this section we show how this can be done. In this section and in Section 2.0 we will make several assumptions. Each assumptions will be discussed and/or relaxed in Section 3.0.

Assume an expert system that distinguishes between two hypotheses, H and ~H. Assume also that the expert system generates belief values that satisfy bel(H)=1-bel(~H). (Call these assumptions A1 and A2 respectively.) Consider Figure 2, which contains two distributions [or densities] P(bel(H)|H) and P(bel(H)|~H). Two thresholds have been set, U and L. Depending on how a user utilizes an expert system, Figure 2 has two different interpretations.

First, the expert system may be utilized to partially automate the inference process. That is, if the expert system outputs very high (low) belief values then the user simply acts under the assumption that H (~H) is true. In this context, U and L can be interpreted as decision thresholds. If bel(H) is greater (less) than U (L), then the user concludes H (~H). Otherwise the user is uncertain (UNC), and proceeds to collect additional evidence. Of course, complete automation occurs when U=L.

Alternatively, the user may view the expert system as a source of evidence. That is, the user combines the expert system output with other data and knowledge to make his or her own inferences. In this context an important question to ask is "How often does the expert system output strong evidence for the correct conclusion?" One standard approach to measuring the strength or diagnosticity of an item of evidence is by a likelihood ratio:

$$LR = \frac{P(bel(H)|H)}{P(bel(H)|\sim H)}.$$

If LR is high (e.g., greater than 10) then the report "bel(H)" is strong evidence for H vs. ~H. If LR is low (e.g., less than .1) then the report "bel(H)" is strong evidence for ~H.

The reason that LR is a standard measure of evidential value is that most theories of rational induction (i.e., proper degrees of belief) recommend the use of Bayes' rule for updating (see Mortimer, 1988). This rule states that a persons relative degree of belief in H vs. ~H, given a new piece of evidence E, should be determined by

$$\frac{P(H|E)}{P(\sim H|E)} \approx \frac{P(E|H)}{P(E|\sim H)} * \frac{P(H)}{P(\sim H)},$$

$$\text{Posterior Odds} \approx \text{LR} * \text{Prior Odds}.$$

U and L in Figure 2 can be interpreted as thresholds of strong evidence. That is, if bel(H) is greater (less) than U (L) then the expert system has output strong evidence for (against) H. If bel(H) is between U and L, then that output does not provide strong evidence in either direction. That is, the user will need to base his or her decision on other factors or be driven by priors.

Consequently, whether the user chooses to utilize the expert system to partially automate inference decisions or as a source of evidence, Figure 2 provides a way of characterizing user/expert system interactions.

Consider P(bel(H)|~H), the distribution of belief values when H is false. Given U and L we can specify three probabilities:

P(bel(H)<L|~H) <-- probability of true negative
P(U>bel(H)>L|~H) <-- probability of uncertain output
P(bel(H)>U|~H) <-- probability of false positive.

Similarly

P(bel(H)<L|H) <-- probability of missed positive
P(U>bel(H)>L|H) <-- probability of uncertain output
P(bel(H)>U|H) <-- probability of true positive.

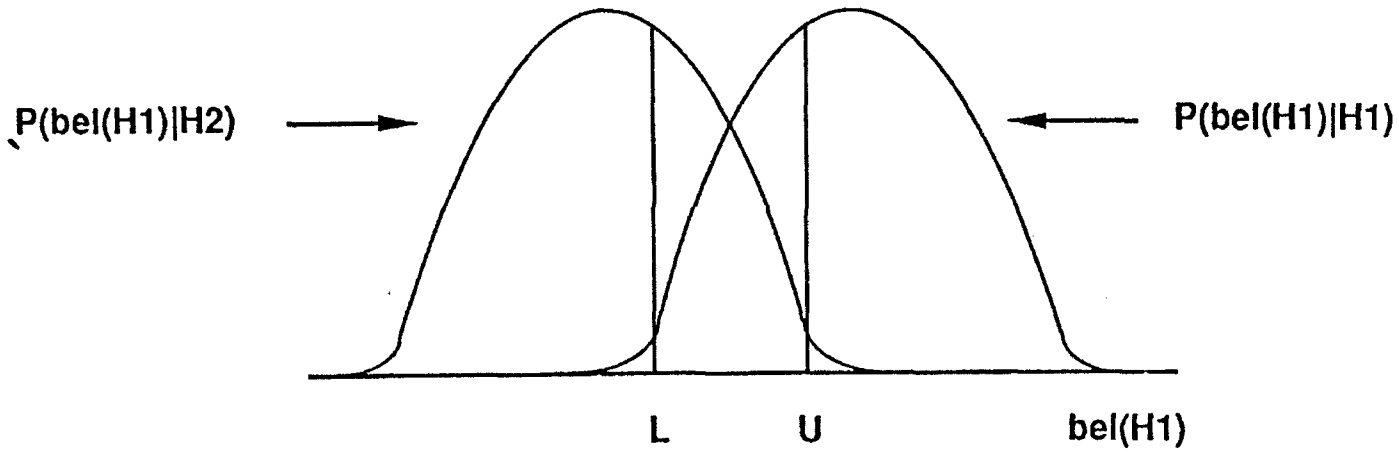


Figure 2: Distributions of Belief Value, bel(H1) for H1-true vs. H2-false

Define P_E to be the probability that the expert system will generate belief values that strongly support the wrong conclusion and P_U to be the probability that the expert system will generate belief values that do not provide strong support for either conclusion. From the above six probabilities we know that

$$P_E = P(bel(H)<L|H)*P(H) + P(bel(H)>U|\sim H)*(1-P(H)) \quad [1]$$

and

$$P_U = P(U>bel(H)>L|H)*P(H) + P(U>bel(H)>L|\sim H)*(1-P(H)) \quad [2]$$

where P(H) is the probability (anticipated relative frequency) of sampling from the H-true distribution.

Together P_E and P_U are two aggregate measures of the usefulness of an expert system. If P_E is relatively high, say .1, then the expert system is generating outputs that strongly support the wrong conclusion about 10% of the time. If P_U is relatively high, say .3, then the expert system is generating useless outputs approximately 30% of the time. From these two numbers we know that $1-P_U-P_E$ is a measure of the proportion of times the expert system will strongly support the correct conclusion.

2.0 USING CLASSICAL STATISTICS

One of the objectives of evaluating an expert system is to assess the extent to which that expert system can help a user to make correct inferences. Although different users will set different U and L thresholds, one can still ask whether it is possible to set thresholds where P_E and P_U are **simultaneously** low. If this cannot be done, then the expert system cannot be very useful in as much as the user must either tolerate a high error rate or a high rate of outputs in the uncertain region.

To estimate the extent to which P_E and P_U can be simultaneously low, it is useful to make several simplifying assumption. They are as follows.

A3) Given each hypothesis, the distributions of belief values are normally distributed.

A4) The distributions of belief values have equal variance.

A5) The U and L thresholds are symmetric. This means that

$$P(bel(H)>U|\sim H) = P(bel(H)<L|H).$$

2.1 Estimating P_E and P_U

Since the thresholds are symmetric and the two normal distributions have equal variance, it follows that P_E and P_U are now independent of the relative frequency of sampling from each distribution. Specifically,

$$P_E = P(bel(H)>U|\sim H) = P(bel(H)<L|H),$$

and

$$P_U = P(U>bel(H)>L|\sim H) = P(U>bel(H)>L|H).$$

In addition, from these assumptions it follows that

$$(M_1-M_2)/s = z(1-P_E) + z(1-P_E-P_U), \qquad [3]$$

where M_1, M_2, s are the means and standard deviation of the two distributions, and z(X) is the z-score for X. From this it can be seen that any procedure for estimating the means and standard deviation of the two distributions will also provide an estimate of P_E and P_U.

Consequently one can specify a straightforward test procedure for evaluating an expert system that discriminates H and ~H. First, identify two representative sources (H-true vs. H-false) of possible test problems. Randomly select problems from each source. Run the expert system against each problem and do a t-test comparison of the results. The t-test analysis will output an estimate of the mean and standard deviation of each distribution, an estimate of the difference between the means of the two distributions and a standard error of the estimate for this difference. From these three estimates, P_E and P_U can be estimated by using equation [3]. An example of this is provided in Section 2.3 below.

2.2 Using P_E and P_U to Determine Sample Size

Although the above procedure is straightforward, we still need to determine the number of test problems required. As it turns out, the P_E and P_U measures can be helpful in making this determination. A standard results from classical statistics (see Hays, 1973, p. 417-422) will be useful here. Namely

$$N = \frac{2[z(1-\alpha) - z(\beta)]^2}{[(M_1-M_2)/s]^2}$$

where N estimates the number of test problems per condition needed to guarantee that if the difference between the two distributions is at least (M_1-M_2/s), then there is at least a $1-\beta$ probability of obtaining significance at the α level in a one-tailed t-test of the null hypothesis of no difference.

Using this equation we can determine a minimum sample size for both groups by specifying the following parameters:

$\max P_E$ - a maximum acceptable error rate,
$\max P_U$ - a maximum acceptable rate of ambiguous results,

α - significance level for t-test
$1-\beta$ - the power of the t-test.

Given these numbers, the minimum sample size for each group is derived as follows:

$$N = \frac{2[z(1-\alpha) - z(\beta)]^2}{[z(1 - \max P_E) + z(1 - \max P_E - \max P_U)]^2}.$$

If $P_E + P_U \leq \max P_E + \max P_U$, then the probability of obtaining a statistically significant difference (at the α level) between the two groups is $\geq (1-\beta)$. As will be illustrated below, using this equation will often result in a minimum sample size that is very small (around 5 tests for each hypothesis in a node).

2.3 An Example

Assume that we have been given the responsibility of testing an expert system with the simple inference network shown in Figure 3. In this inference network there are three evidence items (evid1, evid2 and evid3), one intermediate hypothesis (ihyp1), and one goal hypothesis (ghyp1). Although the analysis does not depend on how belief values are calculated, we note here that bel(ihyp1) is a linear function of bel(evid2) and bel(evid3), and bel(ghyp1) is calculated by performing a relative maximum entropy update given new values for bel(evid1) and bel(ihyp1).

Our first task is to specify a minimum sample size. As evaluators we make the following judgments

(1) An error rate greater than 5% is unacceptable. If the error rate is larger than this, users will simply discard the system.

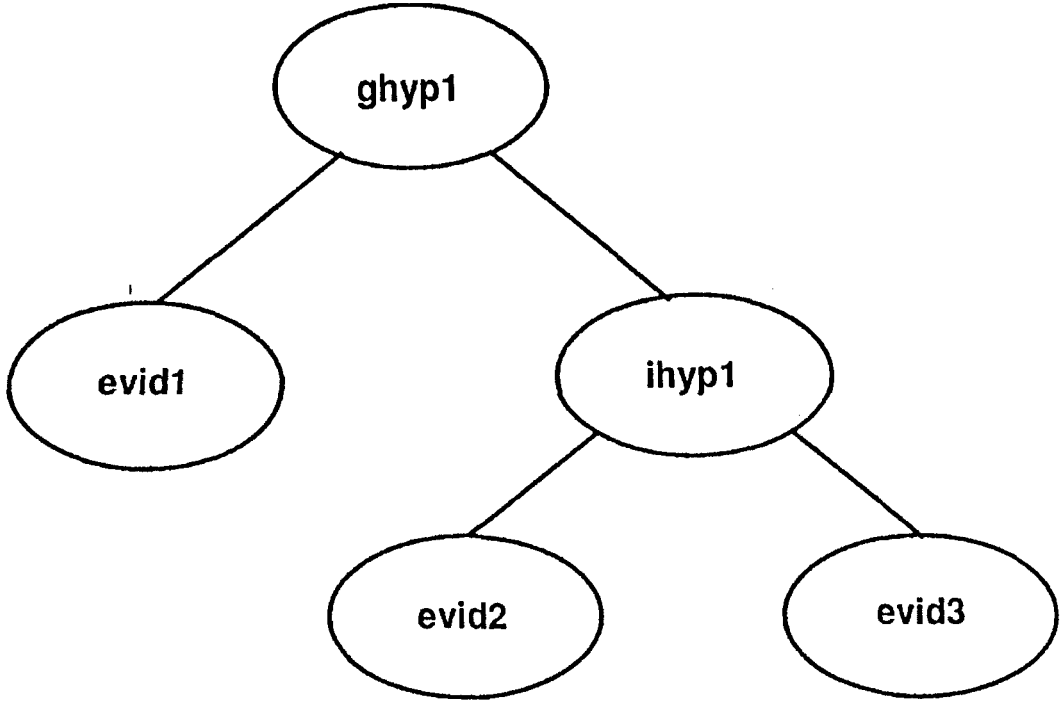


Figure 3: Sample Inference Network

(2) The system should not generate ambiguous results more than 30% of the time. Beyond this level, using the system is considered to be more trouble than its worth.

(3) Set $\alpha = .05$. A level commonly used.

(4) Set $(1 - \beta) = .90$. If indeed the system satisfies (1) and (2) above, then the probability of obtaining one-tailed t-test significance at $p \leq .05$ is .9 or greater.

From these four judgments, we get

$$\max P_U = .30, \quad \max P_E = .05, \quad \alpha = .05, \quad \text{and} \quad (1 - \beta) = .9.$$

This gives us

$$N = \frac{2[z(1-\alpha) - z(\beta)]^2}{[z(1-P_E) + z(1-P_E-P_U)]^2} = \frac{2(1.65 - (-1.28))^2}{[1.65 + .39]^2} \simeq 4.12.$$

So the minimum sample size is approximately four test problems per condition. Even though this seems like a small sample size, if the difference between the two distributions is substantial (i.e., difference between means sufficient to give $P_E+P_U \leq .35$), then there is a 90% chance that this small experiment will generate a t-test result with $p \leq .05$. Consequently, it is unlikely that the expert system, if it satisfies these criteria, will not exhibit at least some difference between the two distributions.

We decide to be "conservative" and let N=8.

After running the 16 randomly selected tests we get the results shown in Table 1. A standard t-test applied to ghyp1 indicates a statistically significant difference between the two sample distributions ($p \leq .00005$). Clearly the expert system has achieved some discrimination between H and ~H.

In addition, we estimate a minimum value for P_U by

$$z(1-\max P_E) + z(1-\max P_E-\text{est}[P_U]) = \text{est}[(M_1-M_2)/s].$$

The observed mean difference is .119, and the estimate of the standard deviation of the distributions is .044. This give us

$$z(.95) + z(.95-\text{est}[P_U]) = \text{est}[M_1-M_2)/s]$$

$$1.65 + z(.95-\text{est}[P_U]) = .119/.044$$

$$z(.95-\text{est}[P_U]) = 1.05$$

$$.95\text{-est}[P_U] = .85$$

$$\text{est}[P_U] = .1$$

TABLE 1
SAMPLE TEST RESULTS

Group (gyp1-true = 1)	Output Belief Values evid1	evid2	evid3	ihyp1	ghyp1
0	.4	.56	.61	.59	.39
0	.4	.43	.28	.36	.33
0	.33	.78	.29	.54	.36
0	.26	.23	.33	.28	.28
0	.48	.26	.32	.29	.34
0	.24	.34	.36	.35	.29
0	.29	.54	.78	.66	.38
0	.48	.48	.34	.41	.37
1	.69	.21	.89	.55	.45
1	.8	.76	.56	.66	.5
1	.44	.89	.48	.69	.42
1	.61	.76	.94	.85	.5
1	.87	.55	.86	.71	.53
1	.59	.56	.4	.48	.41
1	.76	.48	.69	.59	.47
1	.68	.49	.23	.36	.4

Finally, a 90% confidence level for the minimum value of P_U can be estimated by (a) calculating the 90% confidence level for the minimum mean difference and (b) repeating the above procedure. In the case of ghyp1, the observed difference was .119 and the standard error of the estimate of the difference was .022. Consequently, the 90% confidence level for difference is .119 - .022*t(.9, df=14), which is .091. This gives us an "upper bound" on P_U of

$$1.65 + z(.95\text{-est}[P_U]) = .090/.044$$

$$z(.95\text{-est}[P_U]) = .40$$

$$.95\text{-est}[P_U] = .66$$

$$\text{est}[P_U] = .29.$$

This procedure can be repeated for alternative levels of max P_E, from which one can see the tradeoff between P_E and P_U. This is illustrated in Table 2.

TABLE 2
TRADEOFF BETWEEN P_U AND P_E IN SAMPLE PROBLEM

max P_E	est P_U
.1	-.02*
.05	.1
.025	.21
.01	.35
.005	.45
.001	.56

* indicates distributions are sufficiently separated that a single threshold can be set where

$$P(bel(H)>L|\sim H) = P(bel(H)<U|H) \leq \text{max } P_E$$

A similar analysis can be performed for all the nodes in the network. The t-test results for each node in the sample problem are summarized in Table 3. From this table we can draw several conclusions.

Overall, the expert system performs well. As far as the goal node (H vs. ~H) is concerned, a user willing to tolerate a 5% error rate, should find the expert systems advice useful more than 70% of the time, and most likely around 90% of the time. These results do support the evaluation hypothesis that $P_U \leq .3$.

Regarding the other nodes in the network, it seems that most of the discrimination is obtained from evid1, and that the other nodes contribute relatively little to the overall accuracy of the system.

TABLE 3
TEST RESULTS FOR ALL NODES IN SAMPLE PROBLEM
(max P_E set at .05)

node	$M_1 - M_2$	estimate standard deviation	estimate standard error	(max P_E = .05) est P_U	90% C.L. est P_U
evid1	.32	.116	.058	.08	.28
evid2	.135	.197	.099	.78	.86
evid3	.218	.222	.111	.69	.86
ihyp1	.176	.147	.073	.63	.81
ghyp1	.119	.044	.022	.1	.29

3.0 RECONSIDERING THE ASSUMPTIONS

In Sections 1.0 and 2.0 several assumptions were made. They were

A1) The expert system only considers two hypotheses, H and ~H.

A2) The belief values sum to one.

A3) The distributions of belief values are normal.

A4) The distributions of belief values have equal variance.

A5) The thresholds are symmetric.

Each of these assumptions are discussed below. Assumption A3 will be considered last.

3.1 Multiple Hypotheses

If there are just two hypotheses (H and ~H), with belief values that sum to one (A2), then bel(H) completely summarizes both values. When there are more than two hypotheses, this is no longer true. Given a belief value for one hypothesis, the belief values for the other hypotheses can still vary. The implies that for each hypothesis, there is a multivariate distribution of belief values. For instance, if the expert system discriminates three hypotheses H_1, H_2 and H_3; then the output can be characterized as a vector of belief values

$$\underline{b} = \langle bel(H_1), bel(H_2), bel(H_3) \rangle.$$

There are two ways to address the multiple hypothesis case. The first is to perform a multivariate statistical analysis. The thresholds then become hyperplanes in a vector space of possible belief values. For instance, one might set thresholds U_i where for each H_i the decision rule is to select H_i if $bel(H_i) > U_i$. The area defined by $bel(H_i) < U_i$ for all i would then be the uncertain region. P_U is the probability of falling in the uncertain region, while P_E is the probability that for some i, $bel(H_i) > U_i$ occurs when H_i is false. Conceivably one could generalize the evaluation procedure described in Section 2.0 to address this multivariate problem. We have not explored the details of this generalization.

An alternative approach is to do a pairwise comparison of hypotheses. This can proceed as follows. First define a measure, bel_{ij}, that summarizes the relative belief values of the two hypotheses. For example we could set

$$bel_{ij} = bel(H_i)/[bel(H_i)+bel(H_j)],$$

or possibly

$$bel_{ij} = [bel(H_i)-bel(H_j)].$$

Second, determine the minimum sample size required for each pairwise comparison. Third, select a sample size for each H_i that is greater than the maximum of the minimum sample sizes required for each pair comparison involving H_i. Finally, collect the test data and compare each pair of hypotheses as discussed in Section 2.0.

3.2 Belief Values that do not Sum to One.

Many expert systems employ an uncertainty calculus where belief values do not sum to one or where a range of possible values is maintained for each hypothesis. For example, in a Shaferian system of beliefs (Shafer, 1976), bel(H) is often interpreted as the degree to which the existing evidence supports H; where it often occurs that bel(H)+bel(~H)<1.

Conceptually this case is similar to the multiple hypothesis case. For each hypothesis, there is a multivariate distribution of belief values. Consequently, the same techniques apply here. In the case of Shaferian beliefs, for instance, it seems natural that for each pair of hypotheses H_i and H_j, [bel(H_i) - bel(H_j)] effectively summarizes the extent to which the expert system finds evidence that supports H_i vs. H_j.

3.3 Unequal Variance

Assumptions A4, that the two distributions have equal variance, is not essential. The main implication of violating this assumptions is that P_U, but not P_E, now depends on the relative frequency of sampling from the two distributions. This can be seen from equations [1] and [2].

If assumption A4 is not made, then the procedure described in Section 2.0 needs to be modified to (1) estimate the variance of each distribution of belief values separately, and (2) incorporate an estimate of the relative frequency of sampling from each distribution. As long as the thresholds are symmetric, P_E is unaffected by unequal variances. However, P_U will vary; although its value is bounded by $P(U>bel_{ij}>L|H_i)$ and $P(U>bel_{ij}>L|H_j)$, where H_i and H_j are the two hypotheses being compared. A "conservative" estimate for (2) is one that pushes the value for P_U close to its maximum value.

3.4 Nonsymmetric Thresholds

Assumption A5, like A4, is not essential. If A5 is violated, then both P_U and P_E will depend on the relative frequency of sampling from the two distributions. Consequently, if this assumption is violated then the procedure in Section 2.0 must be modified to incorporate a subjective estimate of the relative frequency of sampling from each distribution. Note again that P_U is bounded by $P(U>bel_{ij}>L|H_j)$ and $P(U>bel_{ij}>L|H_j)$, while P_E is bounded by $P(bel_{ij}<L|H_j)$ and $P(bel_{ij}>U|H_j)$. Consequently, a "conservative" estimate of the relative frequency of sampling from the H-true distribution is an estimate that pushes P_U+P_E towards its maximum value.

3.5 Distributions that are not Normal

Assumption A3 is expedient. Although normal distributions are prevalent in nature, there is no guarantee that belief values are always distributed normally. Furthermore, there are procedures for testing the hypothesis that a collection of sample points was generated from a normal distribution. When the test data suggests that the distribution is not normal, then one should consider alternative procedures.

It should be noted, however, that testing an expert system is often an expensive proposition. As a result, the sample size for each distribution is often small (less than ten). Given a small sample size, it is unlikely that a sample distribution will lead to rejecting the assumption of normality, even when the true distribution is not normal. When the normality assumption is incorrect, we are unlikely to detect it.

This leaves us with a quandary - routinely use weaker procedures that make fewer assumptions (viz., nonparametric statistics), or simply assume normality and accept the occasional errors in evaluation that this assumption will entail. In our estimation, for small samples nonparametric procedures (e.g., Lehner, 1989) are too weak to be of much use. Consequently, we recommend the second option.

4.0 Summary and Discussion

In this paper we have examined the use of classical statistical procedures for evaluating the knowledge base of an expert system. Statistical procedures can be used to estimate the relative frequency that an expert system will output belief values that (a) strongly support an incorrect hypothesis, and (b) do not provide strong support for any hypothesis. These procedures can also be

used to estimate the minimum number of test problems required to evaluate an expert system. As it turns out, the required number of test problems per hypothesis is often less than ten.

4.1 Comparison to Other Measures

Other approaches have been proposed to evaluating belief values. As discussed in Levi (1989), for example, probability scoring rules are commonly used to assess the accuracy of probabilistic judgments. The most commonly used scoring rule, initially proposed by Brier (1950), is the mean probability score (MPS). The MPS is simply the average squared error of predictions vs. outcomes. For example, suppose that on three consecutive days a weather forecaster predicts a 20%, 60% and 80% chance of rain. In fact, it rained only on the third day. Then

$$MPS = [(.2-0)^2+(.6-0)^2+(.8-1)^2]/3 = .147.$$

Our approach differs from the use of probability scoring rules in two ways. First, we have focused on measures that have a "behavioral" interpretation. P_U and P_E tell us something about how a user can use an expert system. In contrast, the behavioral implications of "MPS=.147" are unclear. Second, probability scoring rules measure the deviation of outcomes from the absolute belief values. This presupposes that the belief values are probability estimates. Furthermore, it may fail to measure discrimination. Note, for instance, that in Table 1 bel(ghyp1) is almost always less than .5, even when ghyp1 is true. Consequently, the MPS score for this expert system would be very low, even though the expert system effectively discriminates when ghyp1 is true vs. false.

An alternative approach is to estimate the expected cost of incorrect diagnoses (e.g., Levi, 1985; Kalagnanam and Henrion, 1988, Heckerman, 1987). However, this requires that the evaluator (a) set one or more specific thresholds, (b) estimate the cost of different types of errors, and (c) assume the user has no other source of information on the inference problem (else (see Lehner, et.al., 1989) combined user/expert system combination may have a very different pattern of errors than inferences based on the expert system alone). The approach presented in this paper, while similar in spirit, requires fewer assumptions and judgments.

A third approach involves the use of a linear regression analysis to identify any linear relationships between the cues (evidence items) and (a) human expert judgments and (b) the correct diagnosis (see Levi, 1989 for discussion). The argument here is that an expert system is useful only if both (a) and (b) reveal a large nonlinear component, and that expert judgments effectively

predict the nonlinearity between the cues and correct diagnosis. If these conditions are not met, then a complex expert system could be replaced by a much simpler linear model. There is no "added value" in building an expert system. Although this added value approach clearly has merit, the approach presented in this paper addresses an orthogonal issue. Our performance measures do not compare expert system judgments with those of a human expert.

The same is true for other attempts to use statistical tests (e.g., paired t-tests) to compare expert system and human expert judgments (O'Keefe, et.al, 1987). Our comparison is not to human experts, but to "ground truth." Note, however, that this does not mean that "ground truth" cannot be determined by human experts. Indeed, human experts are often required to determine the correct or best answer. However, this determination is usually made post hoc and is based on a great deal of information not available to the decision maker during problem solving.

4.2 Other Statistical Procedures

One final comment. In this paper we have focused only on using commonly used procedures from classical statistics. This was done because most people interested in expert system evaluation have some knowledge of these procedures. However, it is important to note that the P_U and P_E measures do not presuppose these procedures. For instance, recent developments in classical statistics (Efron, 1982) might suggest the use of alternative procedures that would result in a smaller confidence interval around the difference between the means; resulting in smaller confidence level estimates for P_U. Similarly, procedures from Bayesian statistics could also be used to estimate these measures (e.g., Winkler, 1972). Bayesian procedures would also result in smaller intervals, since they would exploit the prior knowledge that the difference between means is unlikely to be negative. We are presently exploring the potential advantages of using alternative procedures.

REFERENCES

Brier, G. (1950) "Verification of forecasts expressed in terms of probability," Monthly Weather Review, 75, 1-3.

Efron, B. (1982) The Bootstrap, the Jackknife and Other Resampling Plans, Philadelphia: Society for Industrial and Applied Mathematics.

Hays, W. (1972) Statistics for the Social Sciences (2nd edition). New York: Holt, Rinehart and Winston.

Heckerman, D. (1987) "An Empirical Comparison of Three Inference Methods," Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence, 158-169.

Kalagnanam, J. and Henrion, M. (1988) "A Comparison of Decision Analysis and Expert Rules for Sequential Diagnosis," Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence, 205-212.

Lehner, P. (1989) "Toward an Empirical Approach to Evaluating the Knowledge Base of an Expert System," IEEE Transactions on Systems, Man and Cybernetics, 19(3), 658-662.

Lehner, P., Mullin, T. and Cohen, M. (1989) "When should a Decision Maker ignore the advice of a Decision Aid," Proceedings of the 1989 Workshop on Uncertainty in AI.

Levi, K. (1985) "A signal detection framework for the evaluation of probabilistic forecasts," Organizational Behavior and Human Performance, 36, 143-166.

Levi, K. (1989) "Expert Systems Should be More Accurate than Human Experts: Evaluation Procedures from Human Judgment and Decision Making," in IEEE Transactions on Systems, Man, and Cybernetics, 19(3), 647-657.

Mortimer, H. (1988) The Logic of Induction (English edition), Chicester, England: Ellis Horwood Limited.

O'Keefe, R., Balci, O. and Smith, E. (1987) "Validating Expert System Performance," IEEE Expert, Winter, 81-90.

Shafer, G. (1976) A Mathematical Theory of Evidence. Princeton University Press.

Winkler, R. (1972) Introduction Bayesian Inference and Decision. New York: Holt, Rinehart and Winston.

KNOWLEDGE-BASED SYSTEMS IN THE ARMY: THE STATE OF THE PRACTICE AND LESSONS LEARNED, WITH IMPLICATIONS FOR TESTING

By:

M.M. Constantine and J.W. Ulvila

Prepared by:

Decision Science Consortium, Inc.
1895 Preston White Drive, Suite 300
Reston, Virginia 22091

Prepared for:

U.S. Army Electronic Proving Ground
Fort Huachuca, Arizona 85613-7110
Attn: STEEP-ET-S (Mr. Robert Harder)

July 1989

This work was supported by the U.S. Army Electronic Proving Ground under Contract #DAEA18-88-C-0028. The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

KNOWLEDGE-BASED SYSTEMS IN THE ARMY: THE STATE OF THE PRACTICE AND LESSONS LEARNED, WITH IMPLICATIONS FOR TESTING*

By
M.M. Constantine
and
J.W. Ulvila

Decision Science Consortium, Inc.

Introduction

During the last few months we have been conducting interviews with people in the Army in the Washington D.C. area who have some experience in developing and testing systems that employ artificial intelligence (particularly expert or knowledge-based systems). The purpose of these interviews is to develop a compendium of "lessons learned" in the areas of developing and testing knowledge-based systems. These efforts are part of the second phase of a Small Business Innovative Research Project sponsored by U.S. Army Electronic Proving Ground.

The objective of the first phase of the project (September 1986 to March 1987) was to develop methods for testing and test evaluation that are appropriate for C^3I systems that employ AI. The goal of Phase II is to develop and demonstrate guidance to aid in the testing of systems that employ AI. This paper supports this goal by surveying and sharing the lessons learned by people in the Army who are developing and testing knowledge-based systems, and by recommending an approach to the testing and evaluation of expert systems.

Many organizations within the Army are involved with prototype systems; some are operational, a few will be fielded, and even fewer are building up a track record. This paper describes the state of the practice, shares the lessons

*To appear in *Proceedings of IJCAI-89 Workshop on Verification, Validation and Testing of Knowledge-Based Systems* 1989.

learned, and offers some advice, based on the lessons learned, for developing and testing operational knowledge-based systems.

The State of the Practice

We began by interviewing Army managers and developers in the Washington D.C. area. The purpose of conducting the interviews was to collect and share the experiences of managers, developers, and testers of knowledge-based systems, and attempt to determine some recommendations for future developers and testers on how to test knowledge-based systems. We are still relatively early in the process of interviewing; the descriptions of Army systems and conclusions drawn so far are based on a sample of interviews.

The interviews were relatively open-ended. The interviewees were asked to provide the following information: a brief description of the system, why the system was considered AI, information on how the requirements for the system were specified, the use of experts (single or multiple) in developing or testing the system, how the system was tested, and what was perceived to be the greatest difficulty or biggest stumbling block.

Knowledge-Based Systems in the Army. At first glance, there appears to be a huge diversity of systems being used by the Army. Knowledge-based systems have been applied to many different tasks, including battle management, hazardous material classification, selection of appropriate contract clauses, in-house support for medical research (human vaccine testing and immunization scheduling), and assigning a rating for psychiatric disability compensation. The knowledge-based systems are based on regulations and knowledge elicited from single or multiple experts in the domain area. The size of the systems

ranges from a number of small "in-house" aids that contain about 50 rules to a very large system that has over 400 procedures, each containing its own set of rules. The systems are designed to run on a variety of machines--personal computers, main frames, Sun workstations, or Symbolics.

However, the systems are all quite similar in their goals. Most, but not all, of the systems performed a support function in an area where the cost of the computer making a mistake was relatively low. Generally, the systems were designed to function as an aid to a person making a decision, rather than as an actual decision maker.

Most of the systems we saw were developed by prototyping. For the most part, requirements documents simply did not exist. Rather, the software either evolved from an existing software program or was created informally to meet a need of a particular department within the organizations. Typically the systems were developed using expert system shells (EXSYS, M-1, CLIPS) or in LISP, which many developers see as a rapid prototyping language. When a system becomes a "final product," some developers see a need to convert it to another language, such as Ada or C. This conversion is a potential problem--one that could be very expensive and even infeasible. Furthermore, the conversion has implications for testing, since some of the testing on the original system will need to be repeated on the final system.

Another problem that surfaced was the use of an expert system as a tutorial for a non-expert. One of the systems designed as a tutorial for non-experts had limited capabilities. The system did not seem to teach the non-expert the skills necessary to become an expert, but rather gave the user "1 year of experience 20 times."

This brings up some of the differences between systems designed for non-experts vs. those designed for experts. Typically both groups interface with the system in a similar manner (answering a series of questions or making selections from a menu), but the expert bases the information for input on his/her knowledge of the problem area, while the non-expert interprets the input data from other sources such as invoices, billing statements, or patient summaries. The difference in end users means that developers and testers need to focus on different aspects of the system, depending on the end user. For example, more emphasis may be needed on the interface, stress testing, and the quality of the explanations when a system is designed for non-experts rather than experts.

As stated earlier, most of these systems developed as prototypes. However, in some cases there were plans to convert from the prototype to an operational system. The requirements documents for these systems generally seem to focus on system integration issues--such as access to large databases or converting to a different shell, a different development language, or a different hardware configuration--and not on issues relating to the expected performance of the software. The lack of documented requirements and the "fuzzy" nature of a knowledge-based system add to the difficulty of testing it.

Testing Knowledge-Based Systems. In asking what it takes to test a knowledge-based system, we received a range of responses. Some think that testing AI software is not any different from testing conventional software ("a rule base is just a structured database"); others believe that testing AI software is very different and requires a multifaceted approach, to include static testing, dynamic testing, multiattribute analysis, acceptance by experts, and questionnaires to users.

Generally, issues concerning testing the expert system were not raised early enough in the development process. Much of the time and resources was spent on knowledge engineering and development. As a result, testing and documentation were crowded out. The emphasis seems to be directed at obtaining a "product" and not at making sure the "product" worked as it should. Often what the system should do was not clearly defined or made explicit, which made testing the system even more difficult. Other difficulties in testing knowledge-based systems were the lack of well-defined standards to test against, the lack of definition of an acceptable level of performance, the complexity of the knowledge base, and unavailability of experts for the development and testing processes. The importance of test and evaluation to the development of an expert system as well as to the ultimate acceptance of the system by the user, sponsors, and organization must be stressed.

Test and evaluation is a critical part of the development cycle. The purpose of test and evaluation is to ensure that the system can be used to solve the particular problem or class of problems for which it was designed. Crucial to the system's ultimate acceptance and use is that the system "prove" itself to the community of users for which it is intended. Typically, an expert is someone we trust to solve a particular problem because that expert has a track record of successful decisions. An expert system needs to prove that it can be trusted. Part of building that trust is providing evidence that the expert system can solve the types of problems it is designed to and can be used by the people who need to use it. A comprehensive and multifaceted approach to test and evaluation is the way to provide that evidence.

An evaluation method for expert systems should provide a means to rigorously test the different parts of the system. The method should allow different

aspects of the system to be weighed differently during various stages in the expert systems life cycle. In addition, the evaluation criteria must be expressed in terms of attributes that can be measured, either objectively or subjectively. Evaluation of an expert system should consist of the following kinds of tests:

Structural Tests of the Knowledge Base. Static or structural testing is concerned with examining the underlying structure of the knowledge base, that is, the logical consistency and logical and functional completeness of the rules. Tests for logical consistency are aimed at finding and correcting redundant rules, subsumed rules, conflicting rules, and unnecessary if conditions. Tests for logical completeness are for finding unreferenced attribute values, illegal attribute values, unreachable conclusions, and deadends in the knowledge base. Functional completeness measures the extent to which the knowledge base addresses all the domain problems that the users have to or need to address.

Content-Specific Tests for the Knowledge Base. In this type of testing we ask the domain expert to make judgments about the accuracy and adequacy of the embedded knowledge. The judgments can be elicited individually with single experts or using group techniques with multiple experts. In this category of tests, the domain expert is asked to assess the accuracy of the facts in the knowledge base, the accuracy of the embedded rules, the acceptability of the knowledge representation scheme, the adequacy of the source, and the modifiability of the knowledge base).

Performance Tests. Performance tests are aimed at determining how well the system carries out its designated functions. These tests can be divided into functions for which "ground truth" answers exists and those for which no ground truth exists. Where ground truth exists, the performance of the system can be compared with a known standard. Where no ground truth exists, we must rely on the judgment of experts to assess the quality of the conclusion. Performance measures should also include measures of response time and time to accomplish the task.

Usability Tests. Usability measures incorporate a number of factors that relate to how well the computer is adapted to the needs of the user. Usability measures can be assessed by observation or opinion survey. Measures of usability that can be observed are the extent of use, manner of use, and features used. Usability measures assessed by opinion survey include confidence in system, ease of use, acceptability of interface, acceptability of results, scope of the application, adequacy and clarity of explanations, and impact on organization.

Additionally, specific tests should be aimed at determining the correctness of the inference engine.

The characteristics or attributes described above can be put into a multiattribute model for evaluating a knowledge-based system. Multiattribute analysis provides an overall framework for evaluation of a system where multiple objectives are important (Ulvila et al., 1987). The multiattribute framework in Figure 1 summarizes the characteristics of the evaluation strategy described above.

As confirmed by the interviews, practice often differs from this paradigm. In practice, the level of testing performed on an expert system seemed to be driven by five factors: (1) the lack of information on testing methods and tools; (2) time constraints; (3) resource constraints; (4) mission criticality; and (5) whether or not the system functions as a decision maker or as an aid to a decision maker. Table 1 describes some of the methods we found that were used to test knowledge-based systems and shows where they fall short in terms of the evaluation criteria specified in the multiattribute model. The strategies are described in terms of level to which they address testing the structure of the knowledge base, the content of the knowledge base, performance, and usability. The level of testing is simply described as one of four values--not addressed (NOT), low, medium, or high. High indicates that a particular attribute was emphasized in the testing strategy, medium indicates it was addressed but not stressed. A ranking of low indicates a low level of attention to testing a particular attribute. NOT indicates that the attribute was not explicitly addressed by the particular testing strategy.

Generally, the different strategies used to test the expert systems possessed the same strengths and weaknesses. All of the testing strategies addressed the content of the knowledge base, the correctness of the reasoning as well as the answer. Testing was performed with the expert and attempts were made to

Figure 1: A MAUA Framework for Integrating Test and Evaluation Criteria

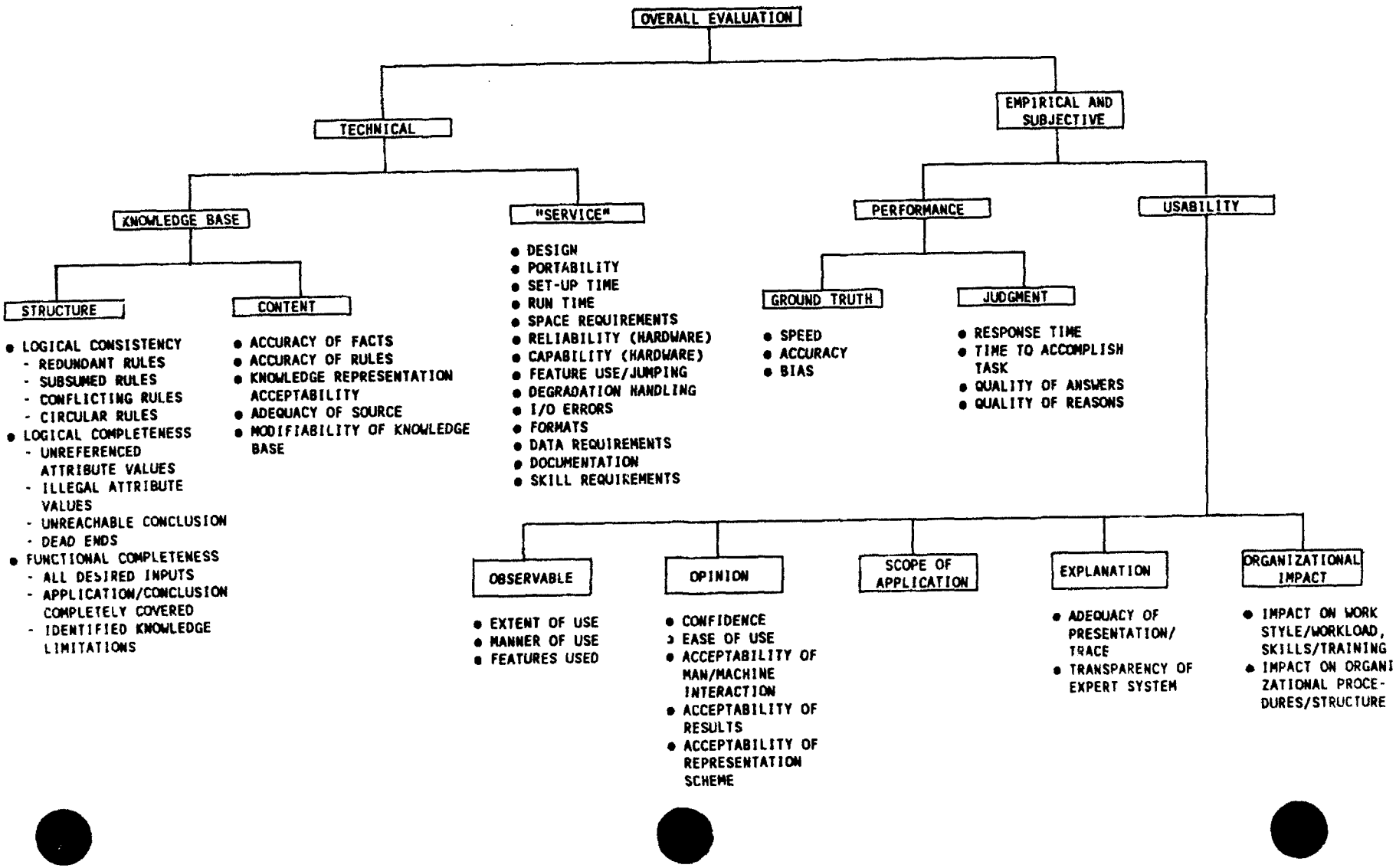


Table 1: Testing Strategies and Level of Testing

Testing Strategy	Level of Testing	
- Show the latest version of the software to the expert; - let the expert use the software; - flag data outside boundaries; - the expert keeps a detailed log of things that go wrong and gives it back to developer.	Structure of KB: Content of KB: Inference Engine: Performance: Correct Answer: Correct Reasoning: Usability: Fit w/Organization:	LOW LOW NOT LOW MED LOW LOW NOT
- Test the system with the expert as it is being developed; - select a sample of test cases based on actual data; - have a panel of experts determine outcomes; - have the computer determine the outcome; - stop testing when the system reaches the same conclusion as the panel of experts 85% of the time; - monitor the system as it is used--establish a track record.	Structure of KB: Content of KB: Inference Engine: Performance: Correct Answer: Correct Reasoning: Usability: Fit w/Organization:	LOW MED NOT MED HIGH LOW LOW NOT
- Select test cases based on actual data (data from a previous year where outcome is already known); - run test cases through the system; - Check the output to ensure that it conforms to Army regulations. Output is correct when the system outcome conforms to a known standard (or regulation specifies what should be).	Structure of KB: Content of KB: Inference Engine: Performance: Correct Answer: Correct Reasoning: Usability: Fit w/Organization:	LOW MED NOT MED HIGH LOW LOW NOT
- Test the interface with eventual users (does it work? does the sequence of questions presented to the user seem logical?); - check the rule base (are rules missing? can some be collapsed? are there rules not being fired? are the rules correct--go back to the expert); - stress-test; - change inputs; - get outputs (are the outputs received in a timely manner? are the outputs reasonable based on the expert's experience in the domain area?); - look at system effectiveness (is the operator happy with the results? is the operator happy with the explanation facility? does the system fit in with the flow of the exercise?).	Structure of KB: Content of KB: Inference Engine: Performance: Correct Answer: Correct Reasoning: Usability: Fit w/Organization:	MED MED LOW MED HIGH LOW MED LOW

Table 1: Testing Strategies and Level of Testing (continued)

Testing Strategy	Level of Testing
- Select test cases that approximate the actual population. The expert determines the output of test cases without using the system. The expert used the system to obtain outputs (in this case, all of the expert's input data were saved); the intended users of this system were non-experts that would be asked to interpret data from summary forms. The saved input data were later used in two ways: 1) to compare the expert's interpretation of data from summary sheets with the non-expert's interpretation; and 2) to re-run the system after changes were made to the rule base. The non-expert ran the system with the same set of test cases. The results of the computer were compared to the results the expert determined without using the system. Testing stopped when the expert and computer agreed 80% of the time.	Structure of KB: LOW Content of KB: MED Inference Engine: NOT Performance: MED Correct Answer: HIGH Correct Reasoning: MED Usability: MED Fit w/Organization: LOW
- Establish test plan--multifaceted approach to include static testing, dynamic testing, multiattribute analysis (for system performance), acceptance by experts (feedback from outside expert as well as "developer" expert), and questionnaires.	Structure of KB: HIGH Content of KB: HIGH Inference Engine: MED Performance: MED Correct Answer: HIGH Correct Reasoning: MED-HIGH Usability: MED-HIGH Fit w/Organization: LOW

ensure the correctness of the embedded knowledge or rules. Many of those interviewed commented that, although experts expected to be available to test the software, they were not. If experts were not available to test the system, the probability of failure was high. Most of the strategies attempted to test the performance of the expert system but this was difficult because the expected performance or measures of performance were not made explicit or documented.

Most of the testing strategies did not sufficiently address the structure of the knowledge base, the correctness of the inference engine, or the usability

of the system. No one addressed the fit with the organization well. Although all of those interviewed examined the structure of the knowledge base. The examination of the structure was extremely difficult and time-consuming because of the lack of automated tools for static analysis. Clearly, the testers and developers would benefit from the availability of automated testing tools. Most of the people realized the importance of structural or static testing but, without automated tools, lack the resources or time to do as much testing as they would like.

Most of the expert systems made use of one of the many expert system shells available on the market, and no testing was specifically aimed at the inference engine of the shell. Most assumed that, when they purchased a shell, the inference engine had already been thoroughly tested. This may not be the case. The literature on the subject indicates that perhaps only one inference engine, CLIPS, has been formally validated. (In their article, "Expert System Verification and Validation", Chris Culbert and Robert Savely of the AI section at NASA/Johnson Space Center describe using conventional testing techniques to validate the inference engine in CLIPS, an expert system shell.) Validating the inference engine of a shell is especially important if a system is performing a critical function and the cost of an error by the system is high.

Because most of the systems were viewed as prototypes (designed to prove the concept), other attributes, such as usability and fit with organization were addressed at a low level or not at all. Lack of information or available tools (questionaires and information on questionnaire design) may also explain the lack of emphasis on these two attributes. Since most of the resources were sunk into development, resources were not available to develop questionnaires.

The testing strategy of changing inputs, obtaining outputs, and asking the expert if the results are reasonable may be appropriate for small, non-autonomous, non-critical, in-house systems (where the cost of an error is small) in environments where the developer and expert work closely together and performance of the system can be continually monitored. However, when the system is performing an important or critical function or where the cost of an error is high, more rigorous and formal testing is necessary.

Lessons Learned

Many of the lessons from the Army's experience in knowledge-based systems relate to requirements and issues that were not raised early enough in the development process. Emphasis was generally placed on developing a product that could be seen, touched, and operated. As a consequence, documentation, testing, and maintenance suffered. Table 2 summarizes the lessons learned.

Table 2: Lessons Learned

(1) Define the Problem

Issues/Question to Ask: Pick a subject that people in the organizations really care about. Why is the system needed? What niche does it fill? What problem does it solve? Make sure the relevant knowledge or information is accessible.

Impact on Testing: Allows testers to know when the problem is solved. Aids in determining if experts exist, who they are, and if they will be available for testing.

(2) Examine Alternative Solutions

Issues/Questions to Ask: AI is one means to solve a problem. Explicitly consider the benefit and costs of alternative approaches. Examine the benefit and cost of a knowledge-based system versus the benefit and cost of a conventional software system versus the benefits and costs associated with no system.

Impact on Testing: Ask if the knowledge-based system performs a function that is important enough in the organization to allocate the necessary resources for its development, testing, and maintenance (some of the costs to consider are the expert's time, test case development, and maintenance of the knowledge base). Will you need to purchase test tools or shells? What will you need to support the testing efforts?

Table 2: Lessons Learned (continued)

(3) Start Early

Issues/Questions to Ask: Many managers and developers of knowledge-based systems feel a need to do more work earlier in the development process. Start answering early the following kinds of questions:

- What do you want the system to do? What are the system's limitations?
- At what frequency will the system be used?
- Is the system important for the user or organization to perform its mission or do its job?
- Who will maintain the system?
- How will the system be tested?
- Do you have the necessary resources to support testing and maintenance?
- What are the milestones along the development path?

Impact on Testing: Work out the logistics of testing--who will test, what will be tested, where will testing be done, when will testing begin and end, how will it be tested? Define the expected performance of the system. Define milestones so that testing at various stages in development provides useful feedback to the development process.

(4) Define the End Users

Issues/Questions to Ask: Who will benefit most from the system? Involve the end user in the design, development, and testing processes.

Impact on Testing: The selected end user has implications in how the interface and explanation facility should be designed and tested. Involving the end users early increases the probability that the users will accept the system.

(5) Commit the Necessary Resources

Issues/Questions to Ask: It is easy to underestimate what it takes to complete the job. Anticipate false starts. Knowledge engineering may take longer than expected.

Impact on Testing: Without the necessary resources, such as the expert's and user's time, the software may not be accepted by the users and the project is likely to fail.

(6) Investigate Shells

Issues/Questions to Ask: Different shells have different strengths; pick the shell best suited for the problem at hand. An expert system shell may not be appropriate given the size or expected performance of a system. Understand the capabilities and limitations of the shell. Tools that help in determining which shell is best suited for a particular application would be beneficial.

Impact on Testing: Tests specifically aimed at the functioning of the inference engine may be important for mission critical systems or systems where the cost of a computer error is high.

(7) Get the Expert's Time

Issues/Question to Ask: Make sure an expert exists. Get a commitment from management to make that expert available. Make sure the expert is willing to cooperate (may lack incentives). When using multiple experts watch for rivalry and differing to higher rank.

Impact on Testing: The expert must be available for test case generation and assessing the adequacy and accuracy of the embedded knowledge. Keep the expert involved in development and testing. Show the expert the results of his knowledge on a prototype (good for self-esteem and may precipitate useful criticism).

Table 2: Lessons Learned (continued)

(8) Define Performance Measures

Issues/Questions to Ask: Explicitly define performance measures. Document the following:

- Computer Aspects: What are the hardware requirements, the space requirements, the machine capacity? What are the data requirements? Does the system need to be ported to other hardware configurations? How fast does the system need to run? What are the formats of the input data?
- The input domain and limitations of the system.
- Expected Performance. Can we measure the system's output against some ground truth or does it requirement judgment? What do we expect in terms of response time, quality of the answer, quality of the reasoning?
- Usability: How much do we expect the system to be used? Does it fit in with the way the organization does business? What features do we expect the system to provide? Which features are most important?
- The Intended User: Who is the intended user? What are the skills required of the user? What level of explanation is required?

Impact on Testing: These issues are critical to the testing process. Defining what is expected from the system allows the tester to know when the system "passes" the tests.

(9) Establish Priorities

Issues/Questions to Ask: Explicitly prioritize the various attributes based on their relative importance.

Impact on Testing: The explicit prioritization provides a useful means to decide which aspects of the system require the most comprehensive testing, especially given time and resource constraints. For example, a system with a large and complex knowledge base designed for a "computer literate" expert should employ a different testing strategy than a system designed for a relatively "computer illiterate" non-expert. In the latter case, testing should emphasize the interface with the user and the sufficiency of the explanations and, in the former case, testing should emphasize the completeness, consistency, accuracy, and adequacy of the knowledge base.

(10) Test the Knowledge Base

Issues/Question to Ask: Comment the rules; indicate their sources. Examine the premises, not just conclusions.

Impact on Testing: Investigate the static test tools to exercise the knowledge base. (There seems to be a strong need for "generic" tools or a library of routines that test the various aspects of a knowledge base. Most of the "testing" currently being performed in the knowledge base is being done by hand and those testing knowledge-based systems would benefit greatly from the availability of such tools.)

(11) Establish a Track Record

Issues/Question to Ask: Keep a library of cases presented to the system. Monitor the performance of the system over time.

Impact on Testing: A library of cases allows for regression testing--re-testing the system after changes have been made to the knowledge base.

(12) Instill Order

Establish a set of AI products (tools, shells) to use in an organization and provide training. Establish training materials that help people select valid AI projects.

To date, successful applications seem to be limited in scope and solve relatively well-defined problems. Some of the people we talked to seemed to think that the future of AI in the Army is in "little modules of AI" as part of larger systems, rather than large AI systems. Generally, prototypes are regarded as useful and valuable for "studying the problem" but do not necessarily lead to operational systems. Prototyping may be nothing more than one step on the way to defining system requirements. To develop operational, usable and testable systems, the Army needs to work harder at answering the difficult questions up front, to explicitly define the scope of the application, the input domain, the limitations of the system, the expected level of performance, the usability requirements, and to rigorously test those aspects of the system. Without rigorous testing, a system cannot be a reliable and useful contributor to the Army's mission.

Acknowledgment

This work was supported by the U.S. Army Electronic Proving Ground under contract number DAEA18-88-C-0028. The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

Reference

Ulvila, J.W., Lehner, P.E., Bresnick, T.A., Chinnis, J.O., Jr. and Gumula, J.D.E. *Testing and evaluating C³I systems that employ artificial intelligence* (Technical Report 87-7). Reston, VA: Decision Science Consortium, Inc., 18 May 1987.

TESTING KNOWLEDGE-BASED SYSTEMS[1,*]

J. W. Ulvila and M. M. Constantine
Decision Science Consortium, Inc.
Reston, Virginia 22091, U.S.A.

L. Adelman
Decision Science Consortium, Inc.
and
George Mason University
Fairfax, Virginia 22030, U.S.A.

ABSTRACT

This paper presents the results of an ongoing project to develop methods for testing knowledge-based systems. Results from four lines of inquiry are presented: a compendium of lessons learned from testing, quality metrics, a multiattribute utility hierarchy, and a Test Technology Program. The results of a personal interview survey of the testing practices of knowledge-based system developers in the U.S. Army are presented. These include: a characterization and critique of common testing strategies, an analysis of factors affecting testing, and a presentation of lessons learned. Metrics for characterizing the quality of knowledge-based systems are described. A comprehensive approach to assessing knowledge-based systems quality based on multiattribute utility analysis is described. The general state of testing for knowledge-based systems and artificial intelligence is described, and a comprehensive Test Technology Program is outlined.

1. INTRODUCTION

We are currently under contract to the U.S. Army Electronic Proving Ground to develop methods for testing systems that employ artificial intelligence. Our two-and-a-half year research and development effort will be completed by September 1990, and this paper summarizes a portion of our work to date. The major focus of our research has been on testing knowledge-based systems or expert systems, and we have found that such systems present some unusual problems for testers due to aspects of their typical development process and due to their intended uses to support human decision making.

The paper is organized as follows. Section 2 presents the results from a personal interview survey of the testing practices of knowledge-based system developers in the U.S. Army. Section 3 presents a multiattribute utility framework for knowledge-based system quality metrics. Section 4 proposes a comprehensive Test Technology Program.

2. COMPENDIUM OF LESSONS LEARNED

This section summarizes the findings from a personal interview survey of the testing practices of knowledge-based system developers in the U.S. Army. The findings include a characterization of common testing strategies, an identification of factors that affect testing, and an expression of lessons learned. More details on these findings can be found in Constantine and Ulvila (1989, 1990).

2.1 CHARACTERIZATION OF COMMON TESTING STRATEGIES

Our survey identified eight strategies that are commonly used to test expert systems.

---

[1]This work was supported by the U.S. Army Electronic Proving Ground under contract number DAEA18-88-C-0028. The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

*To appear in the *Proceedings of the Test Technology Symposium III*, Laurel, MD, 19-21 March 1990.

(1) *Prototype Forever.* The expert receives the latest version of the software and uses it in an actual setting. The expert monitors the system in use and provides feedback on the interface, the explanation facility, and the reasonableness of the system's outputs over time.

(2) *Agreement.* As the system is being developed, it is tested with the expert. When an initial version of the system is complete, a sample of test cases is selected based on actual data. The test cases are given to an expert or a panel of experts who are asked to determine the outcome. The same set of test cases is presented to the system and the system determines the outcome. The system passes the "test" if the system and panel of experts agree on the outcome for some percentage (e.g., 85%) of the test cases. The system is put into use and monitored over time.

(3) *Compliance.* Test cases are selected based on past history. Those cases are presented to the system, the system's performance is compared with the historical results, and appropriate changes are made. Another set of test cases is selected from current data where the outcome is not yet known. These cases are presented to the system and the output is correct if it complies with the relevant regulations.

(4) *Satisfaction.* The developer examines the knowledge base for missing rules, rules that can be collapsed, and rules that are not being fired. The expert subjectively assesses the correctness of the rules, the quality of the explanations, and the quality of the answers. The user assesses his or her ability to interface with the system, the timeliness of the response, the reasonableness of the outputs and explanations, and how the system fits in with the operating environment.

(5) *Case-Dependent.* The developer examines the knowledge base, assesses the effect of adding rules, determines if rules can be combined, and looks for errors. A large sample of test cases is selected that approximates the population of cases the system will receive. The expert assesses the answer to test cases without using the system. Then, the expert uses the system to obtain outputs (all of the expert's actions are saved). The saved data reflecting the expert's actions are analyzed and changes are made to the knowledge base. It is necessary for the expert and system to agree some percentage (e.g., 80%) of the time. The system is then tested with the non-expert users. The non-experts interpret the input data from summary sheets and the differences in data input between the expert and non-expert are examined and appropriate changes are made to the system.

(6) *Organizational Testing.* The interface is iteratively evaluated by the user. Interface evaluation includes an assessment of screen design, feedback message placement, scrolling, features, menu naming, design, and actions. The system is evaluated in a classroom setting by observing the system in use and administering questionnaires. Observers videotape and take notes to assess how both students and instructors use the system in an actual classroom setting. Questionnaires are administered to both students and instructors to gather information regarding features used, perceived usefulness, perceived problems, and general feelings. An experiment, using subjects in an actual classroom environment, is designed to assess the effect of using the system on student performance.

(7) *Field Testing.* Each prototype is tested with past cases from saved actual data. The system is tested in a similar operational environment for several (e.g., 3) months to obtain feedback on system effectiveness and user interface. Appropriate changes are made to the system. The system is then run in parallel with the existing process in the intended operational environment for approximately one year. During the parallel test, assessments are made as to how well the system is meeting the goals stated in the requirements document.

(8) *Multi-Faceted.* The developer performs a comprehensive static analysis of the knowledge base using automated tools. Dynamic testing is performed to test the system with the expert using a comprehensive set of test cases not used in development. Multiattribute analysis is used to obtain subjective measures for system performance. The system is tested with "developer" experts as well as outside experts. Questionnaires are administered to both developer experts and outside experts.

2.2 FACTORS THAT AFFECT TESTING

Generally, the level of testing performed on knowledge-based systems seemed to depend on four factors: information available on testing methods and procedures, time constraints, resource constraints, and characteristics of the development environment (such as formality and accessibility of the developers to the

users). Typically, when the developer and user had a close working relationship, the system was put into use and the developer worked with the user to implement the required software changes on an "as needed" basis.

At some level, all of the common testing strategies addressed correctness of the answer and correctness of the reasoning. All of the systems were judged against some standard, although in many cases that standard was simply a vague notion of "correctness." For example, in agreement testing, the system's output was stamped "correct" if it agreed with the outcomes specified by a panel of experts for 85% of the test cases. In case-dependent testing, great care was taken to select a set of test cases that approximated the [illegible] the system would be expected to handle in an operational environment, but all of the cases used in testing were also used in development. In field testing, the system was judged "correct" if it met a few performance criteria (such as a 30% reduction in downtime) specified in a requirements document.

In one particular project—an intelligent tutoring system—correctness was not as important as clarity of reasoning, usability, or how well the system fit into the intended operational environment. [illegible] organizational testing because strategies like "agreement" and "compliance" did not sufficiently address either usability or fit with the organization.

Many of the testing strategies (prototype forever, agreement, and compliance) focused on the output the system and did not directly address either the structure or content of the knowledge base. Although the satisfaction and case-dependent strategies address the structure and content of the knowledge base, one might argue that they do so insufficiently due to the lack of automated tools for static analysis. Most of the developers realized the importance of structural or static testing, but without automated tools lacked the resources or time to do as much testing as they would have liked. Only the multi-faceted testing strategy used an automated static analysis tool, and it was developed especially for the particular application.

Most of the knowledge-based systems made use of one of the many expert system shells available on the market, and no testing was specifically aimed at the inference engine of the shell. Most developers assumed that, when they purchased a shell, the inference engine had already been tested thoroughly. This may not be the case. The literature indicates that perhaps only one inference engine, CLIPS, has been formally validated (Culbert and Savely, 1988).

The testing strategy of changing inputs, obtaining outputs, and asking the expert if the results are reasonable may be appropriate for small, expendable, non-autonomous, non-critical, in-house systems (where the cost of an error and the cost of the system are extremely low), in environments where the developer and the expert work closely together and the performance of the system is continually monitored. But be prepared; this system is likely to remain a prototype forever. When a knowledge-based system is to be used by a large number of individuals, replace an existing method for solving a particular problem, perform an important or critical function or where the cost of a system error may be high, more rigorous and thorough testing is necessary. One approach for more comprehensive test and evaluation is the multiattribute framework described in Section 3.

2.3 LESSONS LEARNED

The same difficulties were encountered by many knowledge-based system testers. These included:

- issues concerning or affecting testing were not raised early enough in the development process;
- in a resource-constrained environment, it was difficult to test thoroughly;
- the intended purpose or function of the knowledge-based system was not defined clearly or explicitly;
- static analysis tools were unavailable for testing a complex knowledge base;
- experts were unavailable during testing.

Listed below are some of the lessons learned and suggestions for developing testable knowledge-based systems.

[illegible] requirements. It is impossible to test the system without a clear understanding of what the system should do. It is too difficult to do up-front, so it says. Develop a prototype and report on it for what it does mean to generate requirements. (Constantine and Orr [cite] propose an outline for a requirements generation process.)

(2) *Give the knowledge-based system an apprenticeship period in the intended operating environment.* Keep a library of cases presented to the system, observe the system in use, and monitor its performance. An apprenticeship period may be essential for assessing how the system will perform in an operational environment and for finding errors not found in previous testing efforts.

(3) *Instill order.* Establish a set of products (tools, shells) to use, provide training, and build your own tools to support testing if the vendor will not provide them. Automated tools for static analysis are essential for testing a knowledge base thoroughly and should be available to the developers of knowledge-based systems. Since there are no industry standards for knowledge-based systems, specify formats, naming conventions, procedures for commenting rules, and other programming standards for developers in a particular organization to use when building knowledge bases and knowledge-based systems.

To date, successful applications seem to be limited in scope and solve relatively well-defined problems. Many of those interviewed indicated that the future of AI is in "little modules of AI" as part of larger systems, rather than large AI systems. Generally, prototypes were regarded as useful and valuable for "studying the problem" but they did not necessarily lead to operational systems. Prototyping may be nothing more than one step on the way to defining system requirements. To develop operational, usable, and testable systems, developers need to work harder at answering the difficult questions up-front, to define explicitly the scope of the application, the input domain, the limitations of the system, the expected level of performance, the usability requirements, and to rigorously test those aspects of the system. Without rigorous testing, a system cannot be a reliable and useful contributor.

3. A MULTIATTRIBUTE UTILITY HIERARCHY OF QUALITY METRICS

3.1 THE MULTIATTRIBUTE UTILITY HIERARCHY

Multiattribute utility (MAU) analysis is a method that provides a mathematically appropriate procedure for assessing values with multiple effects (see, e.g., Keeney and Raiffa, 1976). As applied to testing knowledge-based systems, MAU provides a framework for representing, in a hierarchy, the many features and criteria appropriate for judging the quality, acceptability, and strengths and weaknesses of knowledge-based systems. The MAU framework presented in Figure 1 shows our proposed list of quality factors for testing knowledge-based system software. More details on MAU analysis, the hierarchy in Figure 1, and the quality metrics are contained in Adelman and Ulvila (forthcoming).

3.2 QUALITY METRICS

The following paragraphs provide brief definitions of the quality metrics that are represented as attributes in the hierarchy.

Logical Consistency. Redundant rules are rules or groups of rules that have essentially the same conditions and conclusions. Redundancy can occur due to duplicate rules or the creation of equivalent rules (rule groups) by wording variations in the names given to variables, or the order in which the rules are processed. Subsumed rules occur when one rule's (or group of rules') meaning is already expressed by another rule (or group of rules) that reaches the same conclusion from similar but less restrictive conditions. Conflicting rules [illegible] combination violates principles of logic (e.g., transitivity). Circular rules are rules that lead one back to an initial (or intermediate) condition instead of a conclusion.

Logical Completeness. Unreferenced attribute values are values on a condition that are not defined; consequently, [illegible] that are outside the acceptable set or range of values for that condition. An attribute that is referenced in a conclusion that cannot be triggered by the rules containing conditions [illegible] connect input conditions with output conclusions.

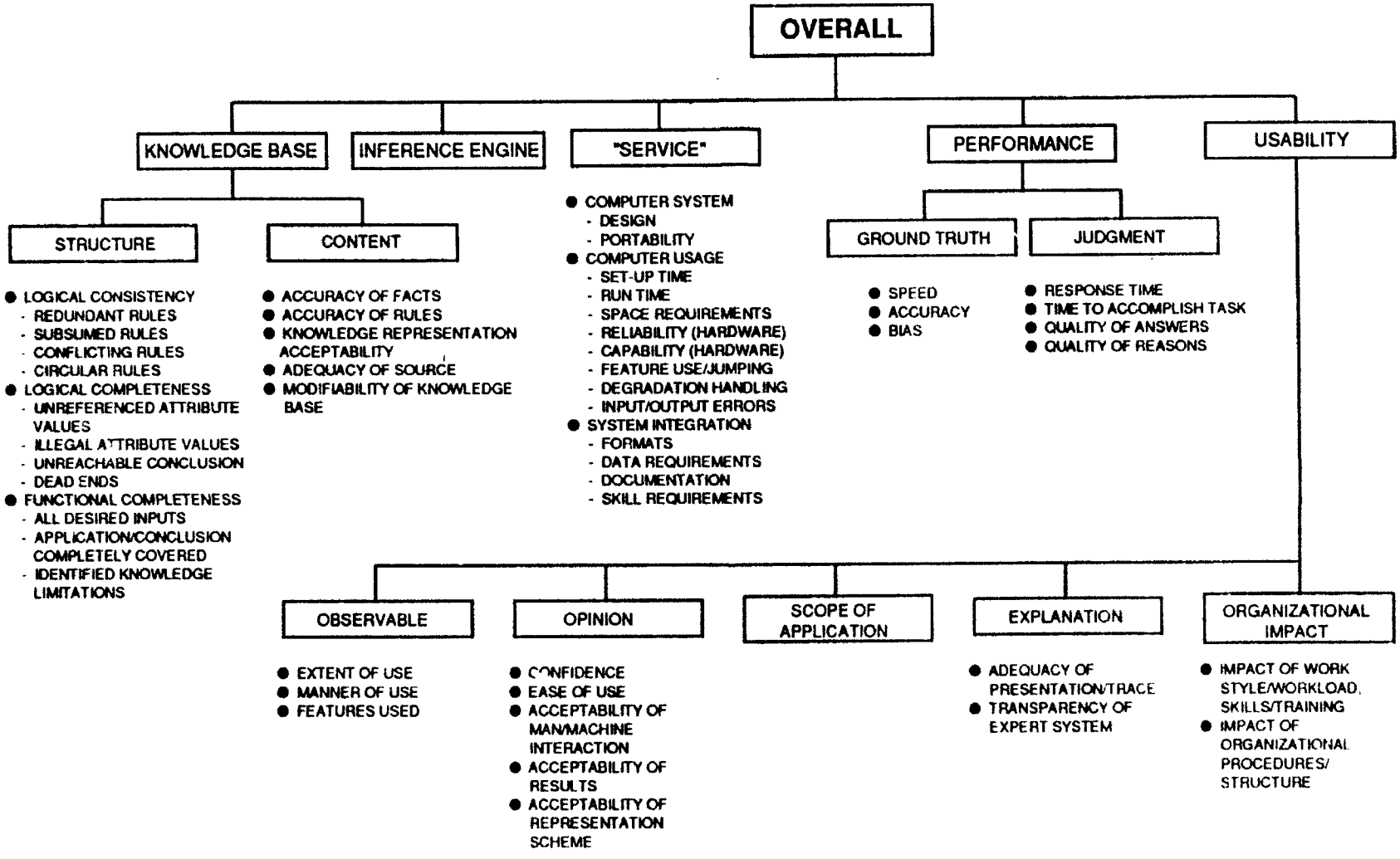


Figure 1. A MAU framework for integrating test and evaluation criteria.

Functional Completeness. All desired inputs: [illegible] need to be addressed. Application/conclusion completely covered: the knowledge base can trigger all conclusions that need to be addressed. Identified knowledge limitations: the rules in the knowledge base can tell the user if input conditions currently being processed cannot be addressed. Analogously, if the expert system is such that a user can specify a conclusion in order to identify the input conditions that would generate it, an expert system that was knowledgeable of its limitations would tell users if a conclusion currently being processed as input could not be addressed.

Knowledge-Base Content. Accuracy of facts: the quality of the unconditional statements in the knowledge base. Accuracy of rules: the quality of the conditional statements in the knowledge base representing expert judgment. Knowledge representation acceptability: whether or not the scheme for representing knowledge is acceptable to other domain experts and knowledge engineers. Adequacy of source: the quality of the persons or documentation used to create the knowledge base. Modifiability of knowledge base: the extent to which the knowledge base can be changed.

Inference Engine: the extent to which the inference engine provides error-free propagation of rules, frames, probabilities, or other representation of knowledge or uncertainties used in the system.

Computer System. Design: the extent to which the expert system runs on the approved computer hardware and operating system and utilizes the preferred complement of equipment and features. Portability: how easily the expert system can be transferred to other computer systems.

Computer Usage. Set-up time: the amount of time required for the computer operator to locate and load the program (if any) and the time to activate the program. Run time: the amount of time required to run the program with a realistic set of input data. Space requirements: the amount of RAM and disk space required by the program. Hardware reliability: the percentage of time the computer system could be expected to be operating effectively. Hardware capability: the computer system's total amount of RAM and disk space. Effect of feature use/jumping: the extent to which moving from various parts of the program causes errors. Degradation: how well the program saves data and analyses and permits continuation after an error, a program or system crash or power outage. Handling input errors: the extent to which the program prohibits a program crash and tells the user what to do after an input mistake.

System Integration. Formats: the extent to which the program uses input and output formats that are consistent with the intended use. Data requirements: the extent to which the program's data requirements are consistent in content, quantity, quality, and timeliness with those available to the intended user organization. Documentation: the adequacy of material regarding the program's use and maintenance. Skill requirements: the extent to which the program can be operated by appropriately skilled individuals.

Performance against Ground Truth. Speed: the amount of time it takes a user working with the expert system to solve representative problem scenarios. Accuracy: the degree of overlap in the distributions of belief values when the hypothesis is true versus false (see Lehner and Ulvila, 1990). Bias: the difference in the proportion of false negatives (hypothesis is true but system says false) to false positives (hypothesis is false, but system say it's true) (see Lehner and Ulvila, 1990).

Judgmental Performance. Response time: the judgments of users regarding the adequacy of the amount of time the expert system takes to react to inputs. Time to accomplish task: the judgments of users regarding the adequacy of the amount of time required to perform the task when using the expert system. Quality of answers: the judgments of users and experts regarding the system's capability. Quality of reasons: the judgments of users and experts regarding the adequacy of the system's justification for its answers.

Observable Usability. Extent of use: how much users employ the expert system to perform the task. Manner of use: the way in which users employ the system and its features, including the procedures to access different modules, the way that intermediate and final outputs are incorporated into the user's results, and the use of interfaces. Features used: the extent to which different aspects of the expert system are employed by users.

Opinions about Usability. Confidence: how confident users feel in taking actions based on work with the expert system. Ease of use: how easy users judge the system is to use after they have completed [illegible]

and become familiar with the system. Acceptability of person/machine interaction process: the extent to which users assess that they and the system are performing the tasks or activities for which they are best suited. Acceptability of results: the users' judgments regarding the adequacy of the system's capability. Acceptability of representation scheme: the users' judgments regarding the adequacy of the system's way of presenting knowledge. Scope of application: the users' judgments regarding the adequacy of the expert system in addressing domain problems. Adequacy of presentation and trace: the users' judgments regarding the acceptability of the system's presentation of its reasoning process. Transparency of expert system: the extent to which the system's reasoning process is clear and understandable to its users.

Organizational Impact. Impact on work style, workload, skills, and training: the judgments of users regarding the impact of the expert system on how they do their job, or the skills and training required to perform it effectively. Impact on organizational procedures and structure: the judgments of users regarding the impact of the expert system on the organization's operations. Input-output: the users' judgments regarding the adequacy of all the expert system's displays except those tracing the reasoning process

3.3 USING THE METRICS AND HIERARCHY IN TESTING

The basic approach to using the MAU hierarchy in testing knowledge-based systems is to assess the performance of the system along each of the metrics, determine the relative importance of the metrics, and combine these factors in an overall characterization of the system being tested. The detailed procedures for applying the method are beyond the scope of this paper (see Adelman and Ulvila, forthcoming), but the following gives some idea of the kinds of techniques that can be used.

Depending on the metric, technical, empirical, or subjective methods might be used. Technical methods are often used in static software testing to characterize the knowledge base and perform tests for logical consistency and completeness. Technical methods can also record the physical parameters of the dynamic operation of the system, such as speed and computer usage. Empirical methods determine objectively questions of fact. These methods could be used to assess performance, content, and observed usability. Subjective methods rely on *judgments for assessments.* These could be the judgments of experts (e.g., on performance) or the judgments of users (e.g., on usability). Specific techniques for eliciting judgments, including the use of questionnaires, are presented in Adelman and Ulvila (forthcoming).

In order to provide overall or summary assessments of the performance of the system being tested, assessments are needed of the relative importance of the various measures or criteria. Where these assessments come from depends on the circumstances of the test. In many cases, it may be appropriate for the Program Manager to indicate the relative importance of the criteria. In other cases, guidance might be given in a requirements document. In some cases, the tester may have to use what information he can gather to infer the intended use of the system and assign relative importance weights himself. Adelman and Ulvila (forthcoming) suggest different sets of weights based on the intended use of the system.

Combining the assessments of the performance of the system against the criteria and the relative importance of the criteria, the tester can make assessments of the system at any level in the hierarchy. Sometimes this assessment is aided by introducing some fictitious systems as reference points (see Ulvila et al., 1987). For example, a "passing score" might be hypothesized for each measure. The system can then be compared measure-by-measure against this hypothetical "passing" system as well as overall (e.g., if the tested system performs better in some areas and worse in others). The hierarchy provides a consistent and structured framework for encoding engineering judgment. The application of the method is facilitated by the computer support system that will be produced in our project.

As a parting thought on the hierarchy, we offer the opinion that an eclectic approach to testing, combining technical, empirical, and subjective methods, is the most effective. As Riedel and Pitz (1986) point out, many people erroneously assume that objective, empirical measurement is the most valid and, therefore, preferred type of data to collect. However, the preference for a particular type of data depends on the relative importance of the criterion being measured by those data. If the system's performance in solving test cases is the most important criterion, then objective empirical data will be the most important type of data to collect. However, if the user's opinion of the expert system is the most important criterion, which is often the case for systems designed to assist experts, then subjective data will be the most important type of data to collect.

Moreover, aggregation of all the test data to make summary assessments of the expert system [illegible] a subjective judgment.

4. A TEST TECHNOLOGY PROGRAM FOR ARTIFICIAL INTELLIGENCE

The last few years have seen an explosion of interest in testing artificial intelligence and knowledge-based systems. As one indication, over 95 papers have appeared on the topic since 1987. However, there is still a long way to go for the testing of artificial intelligence (AI) and knowledge-based systems to reach the level of conventional software testing. This section sketches the components of a test technology program, which could substantially advance the science and practice of testing AI.

(1) *Assess and codify the state-of-the-art in testing.* The AI testing community needs to continue its efforts to get its arms around all the diverse AI testing activities. This requires a comparison and contrast of activities (a) *within* a particular activity area (e.g., various static testing approaches), and (b) *across* areas (e.g., static testing vs. dynamic testing vs. use of experts vs. experiments, etc.). We can easily imagine task forces within and across activity areas, with the result being a major reference work in the field for years to come.

(2) *Develop AI testing laboratories.* There need to be empirical evaluations of alternative testing approaches (and products) within and across activity areas, as well as of completed expert systems and expert system shells in order to assess their adequacy. The Army may want to set up government or commercial laboratories (with no vested interests, e.g., not at product vendors) whose mission is to perform such empirical evaluations. It may be most cost-efficient to have different laboratories specialize in different areas, although this may be premature at this point.

(3) *Package AI testing approaches and products for Army personnel.* A significant effort is required to transfer AI testing technology to Army personnel, and that effort needs to be managed carefully since items 1 and 2 above have yet to be performed. Elements of this include a training program with courses and a place where testers can get hands-on experience, computerized support, and texts.

(4) *Direct efforts toward assessing the value of integrating AI testing into the development process.* It is often argued that such integration will result in better AI systems and reduced development costs, but we are not aware of any empirical studies testing this hypothesis. This could be the first step of a larger project to get testers involved earlier in the development process, to ensure that things such as requirements documents are produced to aid in testing. Managers have to be shown that this involvement is worthwhile, however.

(5) *Direct efforts toward assessing the relative effect of knowledge elicitation techniques, domain experts, knowledge engineers, representation schemes, and problem domains on knowledge-base quality.* It seems quite appropriate for the AI testing community to evaluate the adequacy of the methods that go into building an expert system, not just the finished products (i.e., systems). This would be a major undertaking but especially important if AI is here to stay (e.g., see Adelman, 1989). This would also include the development of testing techniques for "funny logics" (4-valued, non-monotonic, possible-worlds, probabilistic) where appropriate tests do not always exist. Present techniques are focused primarily on rule-based systems (possibly with extensions to frames) and techniques may be needed for other types of systems.

(6) *Develop testing tools.* The five items above are directed at the "science" of testing AI. This item is directed at the "engineering." Tools are needed that get existing methods into the hands of testers. These include:

- static knowledge-based testing tools—for rule-based logics, frame-based logics, and other logics;
- a "requirements" generator—an automated system that will help a tester generate a requirements document from an examination of the system;
- benchmarks and other testing tools for shells and inference engines;
- simple dynamic testing tools (e.g., to keep track of what is going on during the running of the system)—again available for purchase and use;

- comprehensive tools-- such as extensions of the multiattribute utility analysis tool mentioned in Section 3;

- integrative tools to tie other tools together.

REFERENCES

Adelman, L., 1989. Measurement Issues in Knowledge Engineering, IEEE Trans. on Systems, Man, and Cybernetics, SMC-19(3), 483-488.

Adelman, L. and J.W. Ulvila, forthcoming: Testing and Evaluating C^3I Systems that Employ AI: Volume I - Testing and Evaluating Expert Systems (Final Report), Reston, VA: Decision Science Consortium, Inc., 1990.

Constantine, M.M., and J.W. Ulvila, 1989: Knowledge-Based Systems in the Army: The State of the Practice and Lessons Learned, with Implications for Testing, Proceedings of IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge-Based Systems, Detroit, MI: August 19, 1989.

Constantine, M.M. and J.W. Ulvila, 1990: Testing Knowledge-Based Systems: The State of the Practice and Suggestions for Improvement, Expert Systems With Applications.

Culbert, C. and R.T. Savely, 1988: Expert System Verification and Validation, Proceedings of AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems, St. Paul, MN: Aug. 20, 1988.

Keeney, R.L. and H. Raiffa, 1976: Decisions with Multiple Objectives, NY: Wiley.

Lehner, P.E. and J.W. Ulvila, 1990: A Note on the Application of Classical Statistics to Evaluating the Knowledge Base of an Expert System, Reston, VA: Decision Science Consortium, Inc.

Riedel, S.L. and G.F. Pitz, 1986: Utilization-Oriented Evaluation of Decision Support Systems, IEEE Transactions on Systems, Man, and Cybernetics, SMC-16, 980-996.

Ulvila, J.W., P.E. Lehner, T.A. Bresnick, J.O. Chinnis, Jr., and J.D.E. Gumula, 1987: Testing and Evaluating C^3I Systems that Employ Artificial Intelligence (Technical Report 87-7), Falls Church, VA: Decision Science Consortium, Inc.